ureadahead

Resurrection from the dead!

What is ureadhead?

- A system start up tool created by Canonical in 2009 by Scott James Remnant

What is ureadhead?

- A system start up tool created by Canonical in 2009 by Scott James Remnant
  - Who now works for Google!

What is ureadhead?

- A system start up tool created by Canonical in 2009 by Scott James Remnant
  - Who now works for Google!
- Traces files that are opened during boot up

What is ureadhead?

- A system start up tool created by Canonical in 2009 by Scott James Remnant
  - Who now works for Google!
- Traces files that are opened during boot up
- Calls `mincore()` system call to locate memory resident portions of the file

What is ureadhead?

- A system start up tool created by Canonical in 2009 by Scott James Remnant
  - Who now works for Google!
- Traces files that are opened during boot up
- Calls `mincore()` system call to locate memory resident portions of the file
- Creates a “pack” file storing the files and information on what was read

What is ureadhead?

- A system start up tool created by Canonical in 2009 by Scott James Remnant
  - Who now works for Google!
- Traces files that are opened during boot up
- Calls `mincore()` system call to locate memory resident portions of the file
- Creates a “pack” file storing the files and information on what was read
- Subsequent boot ups will use this information to call `readahead()`

Why is this useful?

- When an application execs, it does not get all its memory

Why is this useful?

- When an application execs, it does not get all its memory
- The kernel sets up Virtual Memory Area (VMA) information for the process
  - This is a mapping between the virtual address of the process and where to fill that data

Why is this useful?

- When an application execs, it does not get all its memory
- The kernel sets up Virtual Memory Area (VMA) information for the process
  - This is a mapping between the virtual address of the process and where to fill that data
  - IT DOES NOT FILL IT IMMEDIATELY

Why is this useful?

- When an application execs, it does not get all its memory
- The kernel sets up Virtual Memory Area (VMA) information for the process
  - This is a mapping between the virtual address of the process and where to fill that data
  - IT DOES NOT FILL IT IMMEDIATELY
    - Unless `mlockall()` is used

Why is this useful?

- When an application execs, it does not get all its memory
- The kernel sets up Virtual Memory Area (VMA) information for the process
  - This is a mapping between the virtual address of the process and where to fill that data
  - IT DOES NOT FILL IT IMMEDIATELY
    - Unless `mlockall()` is used
- When the process executes memory that is not filled in yet, it will fault

Why is this useful?

- When an application execs, it does not get all its memory
- The kernel sets up Virtual Memory Area (VMA) information for the process
  - This is a mapping between the virtual address of the process and where to fill that data
  - IT DOES NOT FILL IT IMMEDIATELY
    - Unless `mlockall()` is used
- When the process executes memory that is not filled in yet, it will fault
  - The kernel will then look up the VMA tables and read the memory in

Why is this useful?

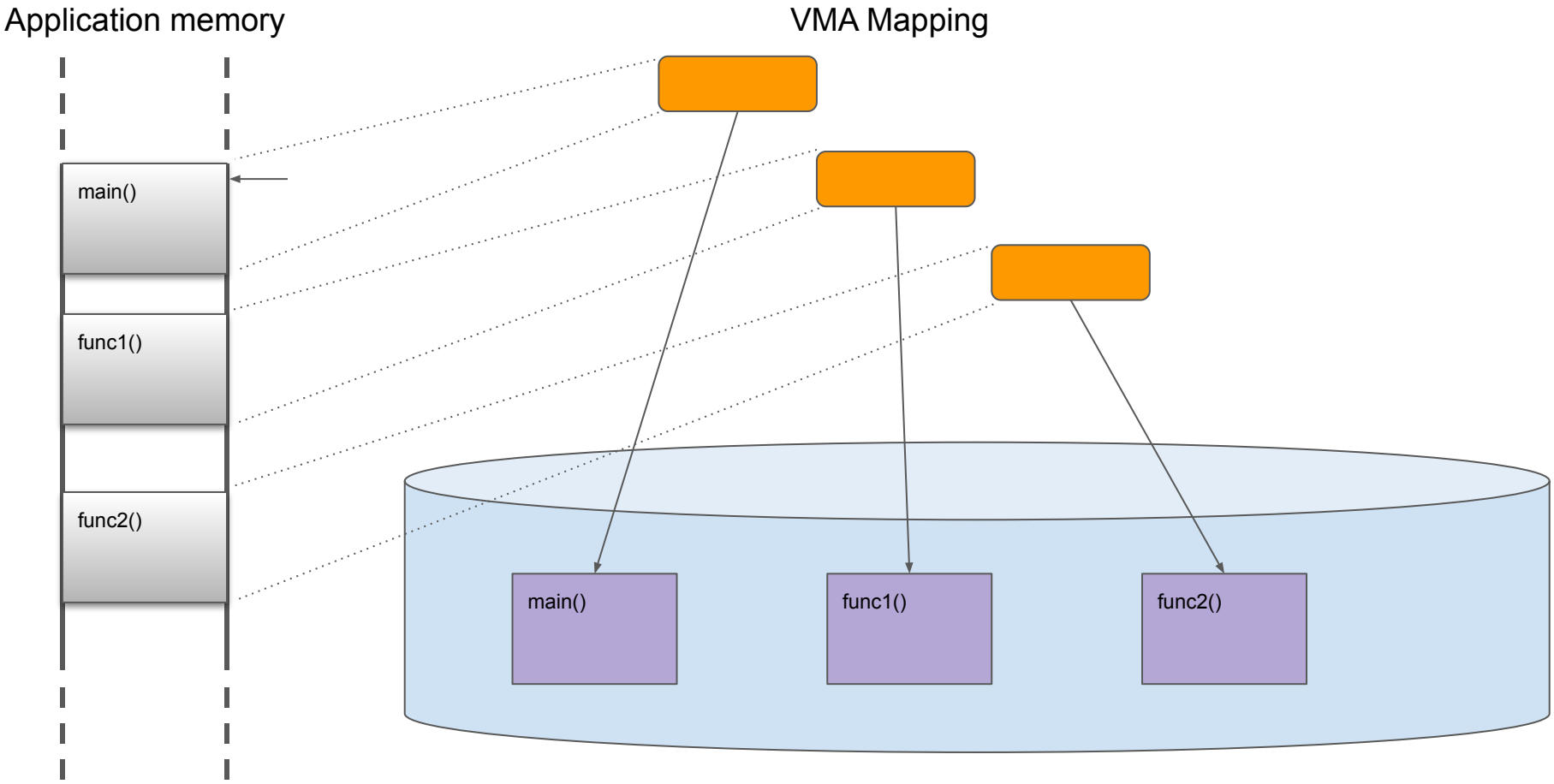
- When an application execs, it does not get all its memory
- The kernel sets up Virtual Memory Area (VMA) information for the process
  - This is a mapping between the virtual address of the process and where to fill that data
  - IT DOES NOT FILL IT IMMEDIATELY
    - Unless `mlockall()` is used
- When the process executes memory that is not filled in yet, it will fault
  - The kernel will then look up the VMA tables and read the memory in
  - If it reads from disk, it is considered a major fault (slow!)

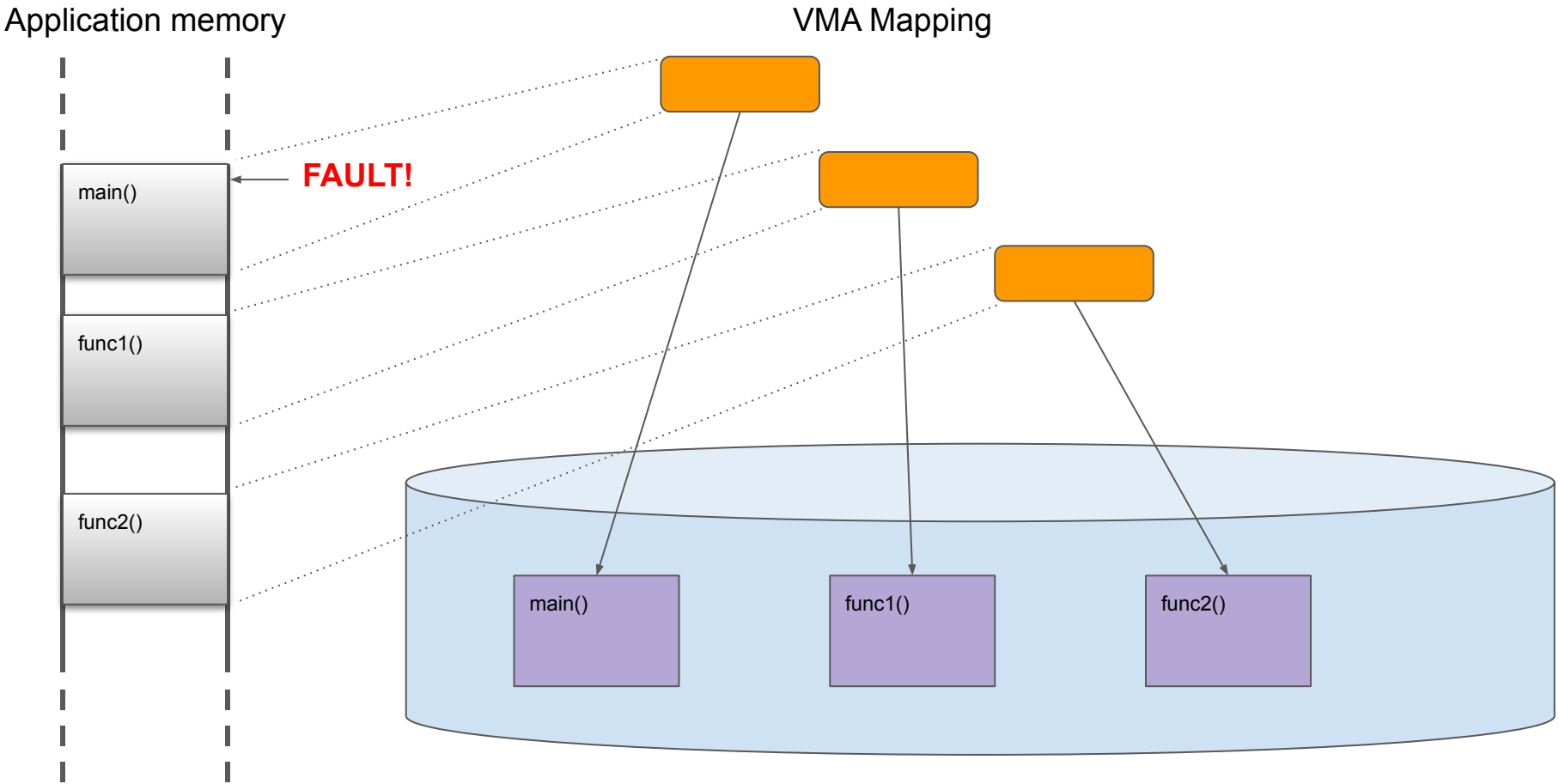
Why is this useful?

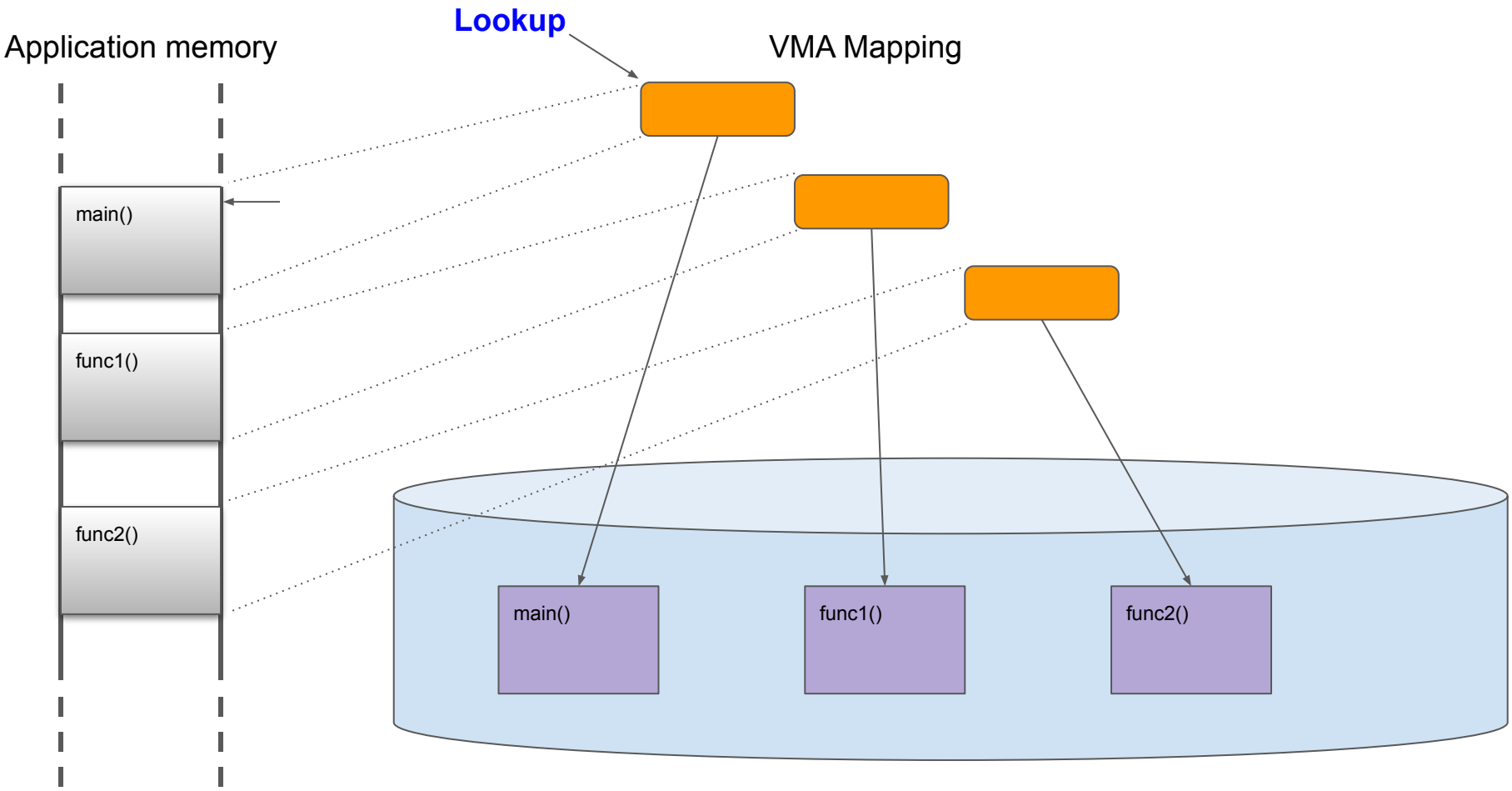
- When an application execs, it does not get all its memory
- The kernel sets up Virtual Memory Area (VMA) information for the process
  - This is a mapping between the virtual address of the process and where to fill that data
  - IT DOES NOT FILL IT IMMEDIATELY
    - Unless `mlockall()` is used
- When the process executes memory that is not filled in yet, it will fault
  - The kernel will then look up the VMA tables and read the memory in
  - If it reads from disk, it is considered a major fault (slow!)
  - If the memory is in the page cache, it is a minor fault (fast!)

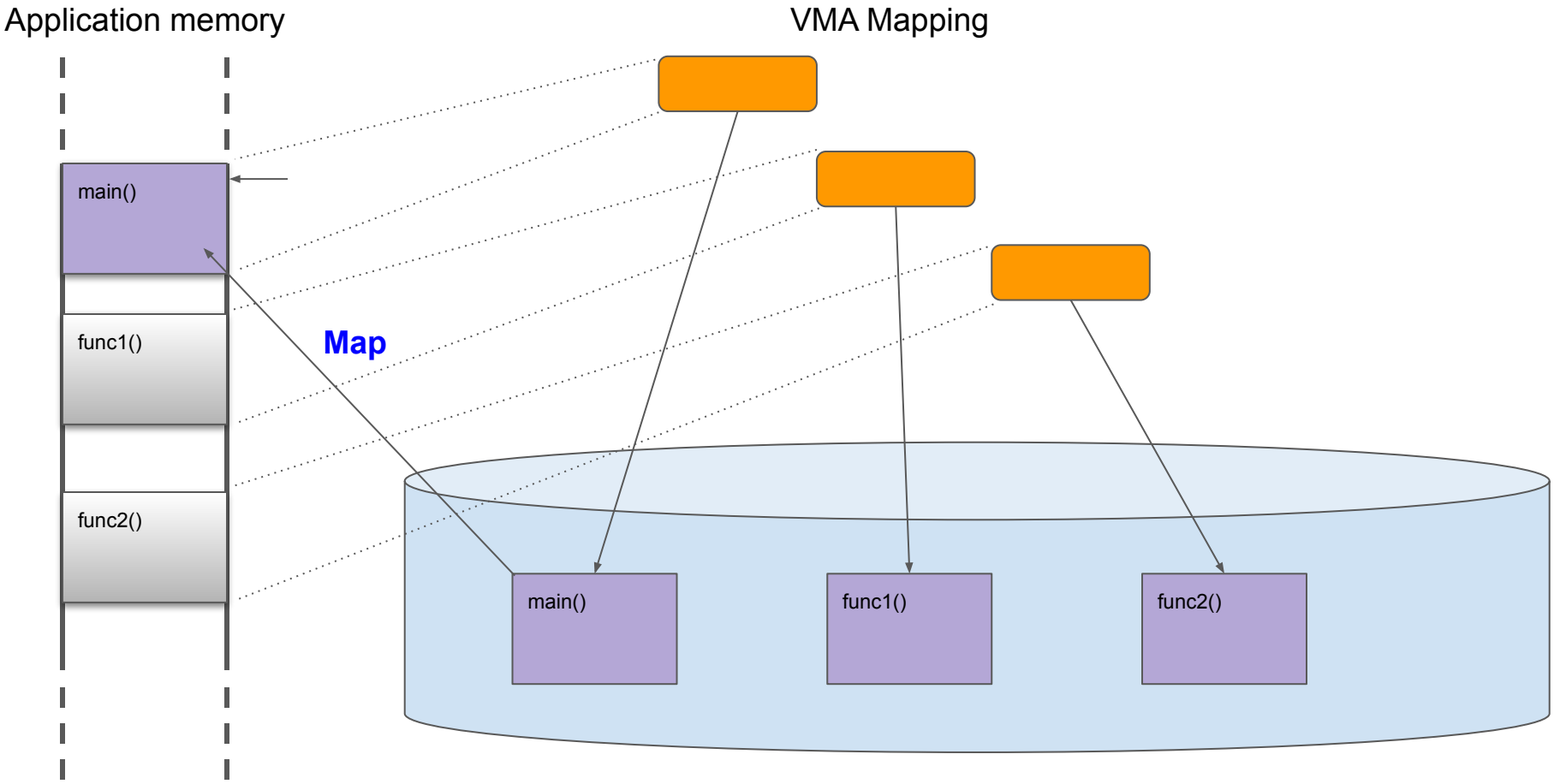
Why is this useful?

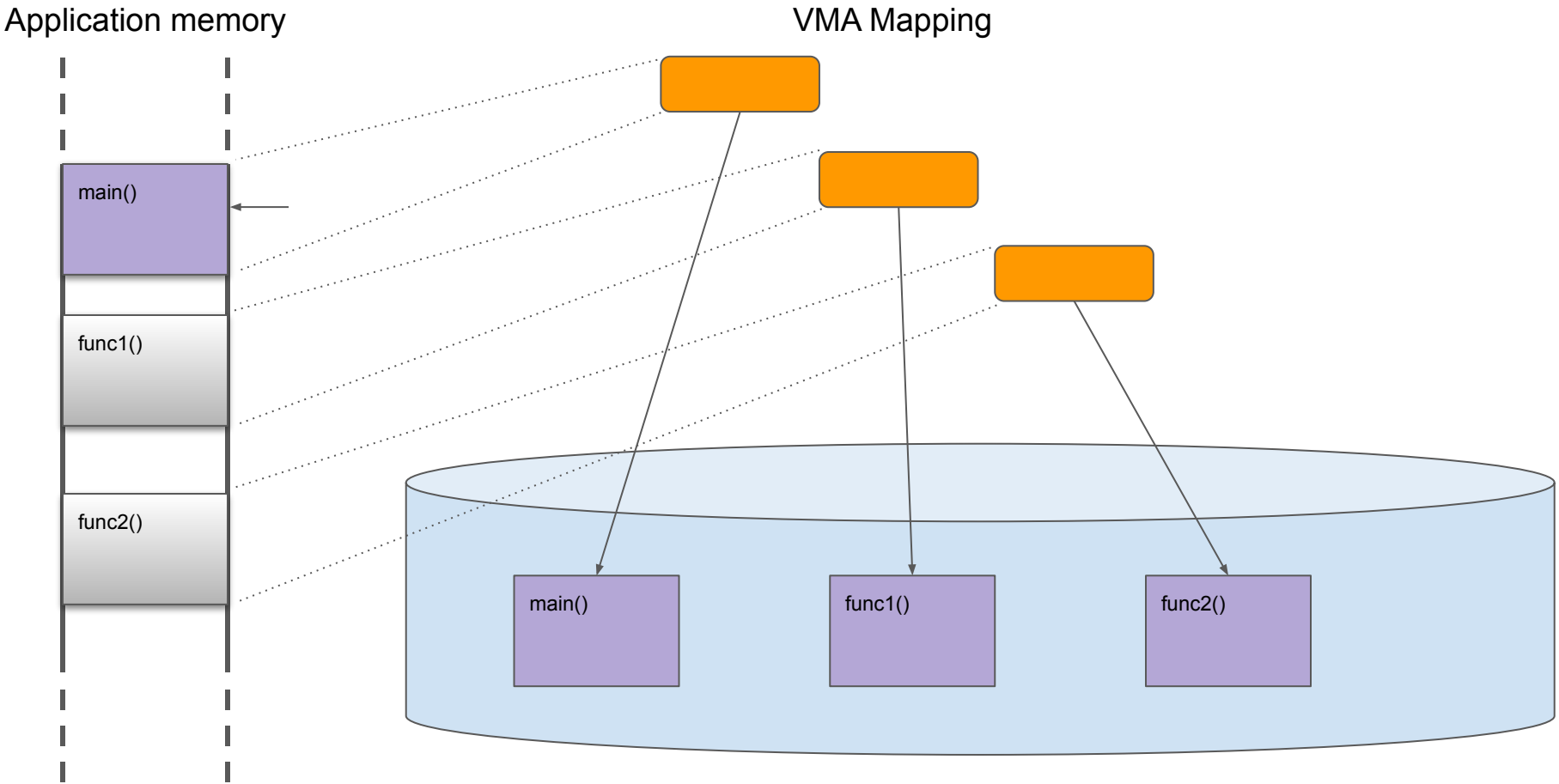
- When an application execs, it does not get all its memory
- The kernel sets up Virtual Memory Area (VMA) information for the process
  - This is a mapping between the virtual address of the process and where to fill that data
  - IT DOES NOT FILL IT IMMEDIATELY
    - Unless `mlockall()` is used
- When the process executes memory that is not filled in yet, it will fault
  - The kernel will then look up the VMA tables and read the memory in
  - If it reads from disk, it is considered a major fault (slow!)
  - If the memory is in the page cache, it is a minor fault (fast!)
- Works for databases that access the same information in a database file

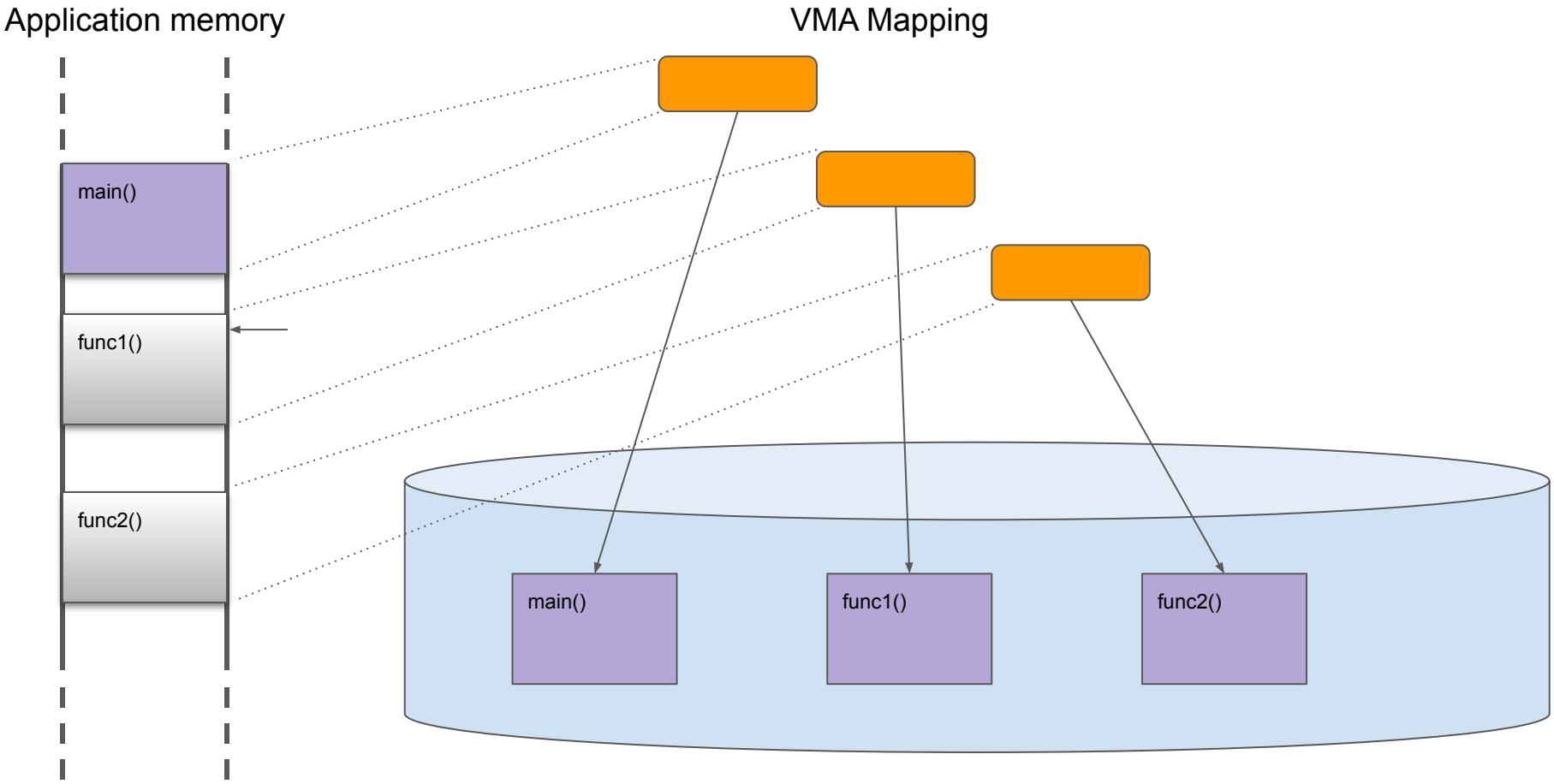

Application memory
VMA Mapping
main()
func1()
func2()
main()
func1()
func2()

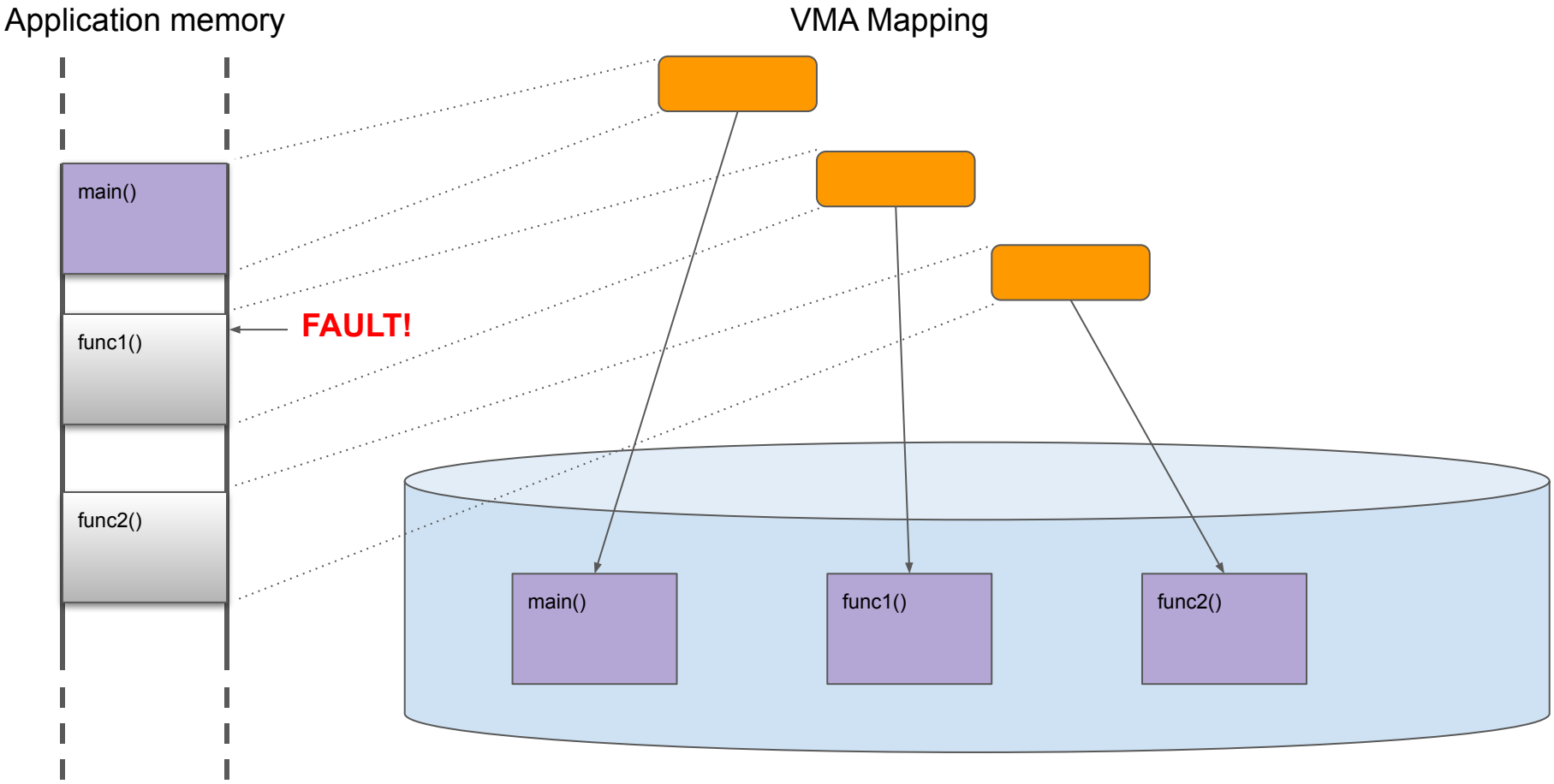

Application memory
VMA Mapping
FAULT!
main()
func1()
func2()
main()
func1()
func2()


Application memory
Lookup
VMA Mapping
main()
func1()
func2()
main()
func1()
func2()


Application memory
VMA Mapping
main()
func1()
func2()
Map
main()
func1()
func2()


Application memory
VMA Mapping
main()
func1()
func2()
main()
func1()
func2()


Application memory
VMA Mapping
main()
func1()
func2()
main()
func1()
func2()


Application memory
VMA Mapping
main()
FAULT!
func1()
func2()
main()
func1()
func2()

Trace Chrome’s page faulting

```
>#  trace-cmd record -p function_graph -l handle_mm_fault -e mm_filemap_add_to_page_cache chrome
>#  mv trace.dat trace-chrome-start.dat
```

Trace Chrome’s page faulting

```
># trace-cmd record -p function_graph -l handle_mm_fault -e mm_filemap_add_to_page_cache chrome
># mv trace.dat trace-chrome-start.dat

># trace-cmd record -p function_graph -l handle_mm_fault -e mm_filemap_add_to_page_cache chrome
># mv trace.dat trace-chrome-second.dat
```

Trace Chrome's page faulting

```
>#  trace-cmd record -p function_graph -l handle_mm_fault -e mm_filemap_add_to_page_cache chrome
>#  mv trace.dat trace-chrome-start.dat

>#  trace-cmd record -p function_graph -l handle_mm_fault -e mm_filemap_add_to_page_cache chrome
>#  mv trace.dat trace-chrome-second.dat

>#  trace-cmd report trace-chrome-start.dat
cpus=8
          chrome-4098  [004] 154614.281855: funcgraph_entry:      + 53.556 us  |  handle_mm_fault();
          chrome-4098  [004] 154614.281982: mm_filemap_add_to_page_cache: dev 254:3 ino e0011 pfn=0x2708df ofs=0 order=0
          chrome-4098  [004] 154614.283043: funcgraph_entry:                   |  handle_mm_fault() {
          chrome-4098  [004] 154614.283083: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x2711c3 ofs=1196032 order=0
          chrome-4098  [004] 154614.283089: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x272183 ofs=1200128 order=0
          chrome-4098  [004] 154614.283093: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x277a68 ofs=1204224 order=0
          chrome-4098  [004] 154614.283098: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x27099b ofs=1208320 order=0
          chrome-4098  [004] 154614.283102: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x270e0d ofs=1212416 order=0
          chrome-4098  [004] 154614.283107: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x270271 ofs=1216512 order=0
          chrome-4098  [004] 154614.283112: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x2709a7 ofs=1220608 order=0
          chrome-4098  [004] 154614.283116: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x270098 ofs=1224704 order=0
          chrome-4098  [004] 154614.283121: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x27068e ofs=1228800 order=0
          chrome-4098  [004] 154614.283126: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x277ce0 ofs=1232896 order=0
          chrome-4098  [004] 154614.283130: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x277b61 ofs=1236992 order=0
          chrome-4098  [004] 154614.283135: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x27100d ofs=1241088 order=0
          chrome-4098  [004] 154614.283139: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x272bc7 ofs=1245184 order=0
          chrome-4098  [004] 154614.283144: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x2705b3 ofs=1249280 order=0
          chrome-4098  [004] 154614.283149: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x276de7 ofs=1253376 order=0
          chrome-4098  [004] 154614.283162: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x271de8 ofs=1257472 order=0
          chrome-4098  [004] 154614.283167: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x2731e0 ofs=1261568 order=0
          chrome-4098  [004] 154614.283531: funcgraph_exit:       ! 489.639 us |  }
```

Trace Chrome's page faulting

```
>#  trace-cmd record -p function_graph -l handle_mm_fault -e mm_filemap_add_to_page_cache chrome
>#  mv trace.dat trace-chrome-start.dat

>#  trace-cmd record -p function_graph -l handle_mm_fault -e mm_filemap_add_to_page_cache chrome
>#  mv trace.dat trace-chrome-second.dat

>#  trace-cmd report trace-chrome-start.dat
cpus=8
          chrome-4098  [004] 154614.281855:                   + 53.556 us   |  handle_mm_fault();
          chrome-4098  [004] 154614.281982: mm_filemap_add_to_page_cache: dev 254:3 ino e0011 pfn=0x2708df ofs=0 order=0
          chrome-4098  [004] 154614.283043: funcgraph_entry:                   |  handle_mm_fault() {
          chrome-4098  [004] 154614.283083: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x2711c3 ofs=1196032 order=0
          chrome-4098  [004] 154614.283089: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x272183 ofs=1200128 order=0
          chrome-4098  [004] 154614.283093: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x277a68 ofs=1204224 order=0
          chrome-4098  [004] 154614.283098: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x27099b ofs=1208320 order=0
          chrome-4098  [004] 154614.283102: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x270e0d ofs=1212416 order=0
          chrome-4098  [004] 154614.283107: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x270271 ofs=1216512 order=0
          chrome-4098  [004] 154614.283112: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x2709a7 ofs=1220608 order=0
          chrome-4098  [004] 154614.283116: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x270098 ofs=1224704 order=0
          chrome-4098  [004] 154614.283121: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x27068e ofs=1228800 order=0
          chrome-4098  [004] 154614.283126: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x277ce0 ofs=1232896 order=0
          chrome-4098  [004] 154614.283130: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x277b61 ofs=1236992 order=0
          chrome-4098  [004] 154614.283135: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x27100d ofs=1241088 order=0
          chrome-4098  [004] 154614.283139: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x272bc7 ofs=1245184 order=0
          chrome-4098  [004] 154614.283144: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x2705b3 ofs=1249280 order=0
          chrome-4098  [004] 154614.283149: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x276de7 ofs=1253376 order=0
          chrome-4098  [004] 154614.283162: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x271de8 ofs=1257472 order=0
          chrome-4098  [004] 154614.283167: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x2731e0 ofs=1261568 order=0
          chrome-4098  [004] 154614.283531: funcgraph_exit:       ! 489.639 us |  }
```

Trace Chrome's page faulting

```
>#  trace-cmd record -p function_graph -l handle_mm_fault -e mm_filemap_add_to_page_cache chrome
>#  mv trace.dat trace-chrome-start.dat

>#  trace-cmd record -p function_graph -l handle_mm_fault -e mm_filemap_add_to_page_cache chrome
>#  mv trace.dat trace-chrome-second.dat

>#  trace-cmd report trace-chrome-start.dat
cpus=8
          chrome-4098  [004] 154614.281855: funcgraph_entry:      + 53.556 us  |  handle_mm_fault();
          chrome-4098  [004] 154614.281982: mm_filemap_add_to_page_cache: dev 254:3 ino e0011 pfn=0x2708df ofs=0 order=0
          chrome-4098  [004] 154614.283043: funcgraph_entry:                   |  handle_mm_fault() {
          chrome-4098  [004] 154614.283083: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x2711c3 ofs=1196032 order=0
          chrome-4098  [004] 154614.283089: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x272183 ofs=1200128 order=0
          chrome-4098  [004] 154614.283093: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x277a68 ofs=1204224 order=0
          chrome-4098  [004] 154614.283098: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x27099b ofs=1208320 order=0
          chrome-4098  [004] 154614.283102: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x270e0d ofs=1212416 order=0
          chrome-4098  [004] 154614.283107: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x270271 ofs=1216512 order=0
          chrome-4098  [004] 154614.283112: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x2709a7 ofs=1220608 order=0
          chrome-4098  [004] 154614.283116: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x270098 ofs=1224704 order=0
          chrome-4098  [004] 154614.283121: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x27068e ofs=1228800 order=0
          chrome-4098  [004] 154614.283126: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x277ce0 ofs=1232896 order=0
          chrome-4098  [004] 154614.283130: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x277b61 ofs=1236992 order=0
          chrome-4098  [004] 154614.283135: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x27100d ofs=1241088 order=0
          chrome-4098  [004] 154614.283139: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x272bc7 ofs=1245184 order=0
          chrome-4098  [004] 154614.283144: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x2705b3 ofs=1249280 order=0
          chrome-4098  [004] 154614.283149: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x276de7 ofs=1253376 order=0
          chrome-4098  [004] 154614.283162: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x271de8 ofs=1257472 order=0
          chrome-4098  [004] 154614.283167: mm_filemap_add_to_page_cache: dev 254:3 i
          chrome-4098  [004] 154614.283531: funcgraph_exit:       ! 489.639 us |  }
```

Major fault

Using: libtracecmd

```
int main (int argc, char **argv)
{
      struct tracecmd_input *handle;
      struct data data = {};

      handle = tracecmd_open(argv[1], 0);

      tracecmd_follow_event(handle, "ftrace", "funcgraph_exit",
                            func_graph_exit, &data);
      tracecmd_follow_event(handle, "filemap", "mm_filemap_add_to_page_cache",
                            mm_filemap, &data);
      tracecmd_iterate_events(handle, NULL, 0, NULL, NULL);
      tracecmd_close(handle);

      printf("Page faults:         %lld\n", data.nr_page_faults);
      printf("Page fault time:     ");
      print_time(data.page_fault_time);
      printf("file mapping count:  %lld\n", data.nr_filemaps);

      return 0;
}
```

Using: libtracecmd

```
int main (int argc, char **argv)
{
      struct tracecmd_input *handle;
      struct data data = {};

      handle = tracecmd_open(argv[1], 0);

      tracecmd_follow_event(handle, "ftrace", "funcgraph_exit",
                            func_graph_exit, &data);
      tracecmd_follow_event(handle, "filemap", "mm_filemap_add_to_page_cache",
                            mm_filemap, &data);
      tracecmd_iterate_events(handle, NULL, 0, NULL, NULL);
      tracecmd_close(handle);

      printf("Page faults:         %lld\n", data.nr_page_faults);
      printf("Page fault time:     ");
      print_time(data.page_fault_time);
      printf("file mapping count:  %lld\n", data.nr_filemaps);

      return 0;
}
```

Using: libtracecmd

```
int main (int argc, char **argv)
{
      struct tracecmd_input *handle;
      struct data data = {};

      handle = tracecmd_open(argv[1], 0);

      tracecmd_follow_event(handle, "ftrace", "funcgraph_exit",
                            func_graph_exit, &data);
      tracecmd_follow_event(handle, "filemap", "mm_filemap_add_to_page_cache",
                            mm_filemap, &data);
      tracecmd_iterate_events(handle, NULL, 0, NULL, NULL);
      tracecmd_close(handle);

      printf("Page faults:         %lld\n", data.nr_page_faults);
      printf("Page fault time:     ");
      print_time(data.page_fault_time);
      printf("file mapping count:  %lld\n", data.nr_filemaps);

      return 0;
}
```

Using: libtracecmd

```
int main (int argc, char **argv)
{
      struct tracecmd_input *handle;
      struct data data = {};

      handle = tracecmd_open(argv[1], 0);

      tracecmd_follow_event(handle, "ftrace", "funcgraph_exit",
                            func_graph_exit, &data);
      tracecmd_follow_event(handle, "filemap", "mm_filemap_add_to_page_cache",
                            mm_filemap, &data);
      tracecmd_iterate_events(handle, NULL, 0, NULL, NULL);
      tracecmd_close(handle);

      printf("Page faults:         %lld\n", data.nr_page_faults);
      printf("Page fault time:     ");
      print_time(data.page_fault_time);
      printf("file mapping count:  %lld\n", data.nr_filemaps);

      return 0;
}
```

```
static int func_graph_exit(struct tracecmd_input *handle, struct tep_event *event,
                           struct tep_record *record, int cpu, void *data)
{
      struct tep_handle *tep = tracecmd_get_tep(handle);
      static struct tep_format_field *func_field;
      static struct tep_format_field *call_field;
      static struct tep_format_field *ret_field;
      unsigned long long calltime, rettime, val;
      struct data *d = data;
      const char *func;

      if (!func_field) {
            func_field = tep_find_field(event, "func");
            call_field = tep_find_field(event, "calltime");
            ret_field = tep_find_field(event, "rettime");
      }

      tep_read_number_field(func_field, record->data, &val);
      func = tep_find_function(tep, val);

      tep_read_number_field(call_field, record->data, &calltime);
      tep_read_number_field(ret_field, record->data, &rettime);

      if (strcmp(func, "handle_mm_fault") == 0) {
            d->nr_page_faults++;
            d->page_fault_time += rettime - calltime;
      }
      return 0;
}
```

```
static int func_graph_exit(struct tracecmd_input *handle, struct tep_event *event,
                           struct tep_record *record, int cpu, void *data)
{
      struct tep_handle *tep = tracecmd_get_tep(handle);
      static struct tep_format_field *func_field;
      static struct tep_format_field *call_field;
      static struct tep_format_field *ret_field;
      unsigned long long calltime, rettime, val;
      struct data *d = data;
      const char *func;

      if (!func_field) {
            func_field = tep_find_field(event, "func");
            call_field = tep_find_field(event, "calltime");
            ret_field = tep_find_field(event, "rettime");
      }

      tep_read_number_field(func_field, record->data, &val);
      func = tep_find_function(tep, val);

      tep_read_number_field(call_field, record->data, &calltime);
      tep_read_number_field(ret_field, record->data, &rettime);

      if (strcmp(func, "handle_mm_fault") == 0) {
            d->nr_page_faults++;
            d->page_fault_time += rettime - calltime;
      }
      return 0;
}
```

```
static int func_graph_exit(struct tracecmd_input *handle, struct tep_event *event,
                           struct tep_record *record, int cpu, void *data)
{
      struct tep_handle *tep = tracecmd_get_tep(handle);
      static struct tep_format_field *func_field;
      static struct tep_format_field *call_field;
      static struct tep_format_field *ret_field;
      unsigned long long calltime, rettime, val;
      struct data *d = data;
      const char *func;

      if (!func_field) {
            func_field = tep_find_field(event, "func");
            call_field = tep_find_field(event, "calltime");
            ret_field = tep_find_field(event, "rettime");
      }

      tep_read_number_field(func_field, record->data, &val);
      func = tep_find_function(tep, val);

      tep_read_number_field(call_field, record->data, &calltime);
      tep_read_number_field(ret_field, record->data, &rettime);

      if (strcmp(func, "handle_mm_fault") == 0) {
            d->nr_page_faults++;
            d->page_fault_time += rettime - calltime;
      }
      return 0;
}
```

```
static int func_graph_exit(struct tracecmd_input *handle, struct tep_event *event,
                           struct tep_record *record, int cpu, void *data)
{
      struct tep_handle *tep = tracecmd_get_tep(handle);
      static struct tep_format_field *func_field;
      static struct tep_format_field *call_field;
      static struct tep_format_field *ret_field;
      unsigned long long calltime, rettime, val;
      struct data *d = data;
      const char *func;

      if (!func_field) {
            func_field = tep_find_field(event, "func");
            call_field = tep_find_field(event, "calltime");
            ret_field = tep_find_field(event, "rettime");
      }

      tep_read_number_field(func_field, record->data, &val);
      func = tep_find_function(tep, val);

      tep_read_number_field(call_field, record->data, &calltime);
      tep_read_number_field(ret_field, record->data, &rettime);

      if (strcmp(func, "handle_mm_fault") == 0) {
            d->nr_page_faults++;
            d->page_fault_time += rettime - calltime;
      }
      return 0;
}
```

```
static int func_graph_exit(struct tracecmd_input *handle, struct tep_event *event,
                    struct tep_record *record, int cpu, void *data)
{
      struct tep_handle *tep = tracecmd_get_tep(handle);
      static struct tep_format_field *func_field;
      static struct tep_format_field *call_field;
      static struct tep_format_field *ret_field;
      unsigned long long calltime, rettime, val;
      struct data *d = data;
      const char *func;

      if (!func_field) {
            func_field = tep_find_field(event, "func");
            call_field = tep_find_field(event, "calltime");
            ret_field = tep_find_field(event, "rettime");
      }

      tep_read_number_field(func_field, record->data, &val);
      func = tep_find_function(tep, val);

      tep_read_number_field(call_field, record->data, &calltime);
      tep_read_number_field(ret_field, record->data, &rettime);

      if (strcmp(func, "handle_mm_fault") == 0) {
            d->nr_page_faults++;
            d->page_fault_time += rettime - calltime;
      }
      return 0;
}
```

```
static int func_graph_exit(struct tracecmd_input *handle, struct tep_event *event,
                           struct tep_record *record, int cpu, void *data)
{
      struct tep_handle *tep = tracecmd_get_tep(handle);
      static struct tep_format_field *func_field;
      static struct tep_format_field *call_field;
      static struct tep_format_field *ret_field;
      unsigned long long calltime, rettime, val;
      struct data *d = data;
      const char *func;

      if (!func_field) {
            func_field = tep_find_field(event, "func");
            call_field = tep_find_field(event, "calltime");
            ret_field = tep_find_field(event, "rettime");
      }

      tep_read_number_field(func_field, record->data, &val);
      func = tep_find_function(tep, val);

      tep_read_number_field(call_field, record->data, &calltime);
      tep_read_number_field(ret_field, record->data, &rettime);

      if (strcmp(func, "handle_mm_fault") == 0) {
            d->nr_page_faults++;
            d->page_fault_time += rettime - calltime;
      }
      return 0;
}
```

```
static int func_graph_exit(struct tracecmd_input *handle, struct tep_event *event,
                           struct tep_record *record, int cpu, void *data)
{
      struct tep_handle *tep = tracecmd_get_tep(handle);
      static struct tep_format_field *func_field;
      static struct tep_format_field *call_field;
      static struct tep_format_field *ret_field;
      unsigned long long calltime, rettime, val;
      struct data *d = data;
      const char *func;

      if (!func_field) {
            func_field = tep_find_field(event, "func");
            call_field = tep_find_field(event, "calltime");
            ret_field = tep_find_field(event, "rettime");
      }

      tep_read_number_field(func_field, record->data, &val);
      func = tep_find_function(tep, val);

      tep_read_number_field(call_field, record->data, &calltime);
      tep_read_number_field(ret_field, record->data, &rettime);

      if (strcmp(func, "handle_mm_fault") == 0) {
            d->nr_page_faults++;
            d->page_fault_time += rettime - calltime;
      }
      return 0;
}
```

```
static int mm_filemap(struct tracecmd_input *handle, struct tep_event *event,
                      struct tep_record *record, int cpu, void *data)
{
      struct data *d = data;

      d->nr_filemaps++;
      return 0;
}
```

Using: libtracecmd

```
int main (int argc, char **argv)
{
      struct tracecmd_input *handle;
      struct data data = {};

      handle = tracecmd_open(argv[1], 0);

      tracecmd_follow_event(handle, "ftrace", "funcgraph_exit",
                            func_graph_exit, &data);
      tracecmd_follow_event(handle, "filemap", "mm_filemap_add_to_page_cache",
                            mm_filemap, &data);
      tracecmd_iterate_events(handle, NULL, 0, NULL, NULL);
      tracecmd_close(handle);

      printf("Page faults:         %lld\n", data.nr_page_faults);
      printf("Page fault time:     ");
      print_time(data.page_fault_time);
      printf("file mapping count:  %lld\n", data.nr_filemaps);

      return 0;
}
```

Using: libtracecmd

```
#define NS_PER_SEC 1000000000ULL

static void print_time(unsigned long long time)
{
	unsigned long long secs;
	unsigned long long usecs;

	secs = time / NS_PER_SEC;
	usecs = time - (secs * NS_PER_SEC);
	usecs /= 1000;
	printf("%lld.%06lld\n", secs, usecs);
}
```

Using: libtracecmd

```
#define NS_PER_SEC 1000000000ULL

static void print_time(unsigned long long time)
{
	unsigned long long secs;
	unsigned long long usecs;

	secs = time / NS_PER_SEC;
	usecs = time - (secs * NS_PER_SEC);
	usecs /= 1000;
	printf("%lld.%06lld\n", secs, usecs);
}
```

Code at: https://rostedt.org/code/cnt-page-faults.c

Compile with: `gcc -o cnt-page-faults cnt-page-faults.c `pkg-config --cflags --libs libtracecmd``

Using: libtracecmd

```
>$ ./cnt-page-faults trace-chrome-start.dat

Page faults:         100008
Page fault time:     1.984041
file mapping count:  74002

>$ ./cnt-page-faults trace-chrome-second.dat

Page faults:         90018
Page fault time:     0.777085
file mapping count:  183
```

ureadahead records what is read

- The first boot traces files opened

ureadahead records what is read

- The first boot traces files opened
  - After the trace it reads the memory that is mapped

ureadahead records what is read

- The first boot traces files opened
  - After the trace it reads the memory that is mapped
  - Creates a “pack” file

ureadahead records what is read

- The first boot traces files opened
  - After the trace it reads the memory that is mapped
  - Creates a “pack” file
- The next boot reads the “pack” file

ureadahead records what is read

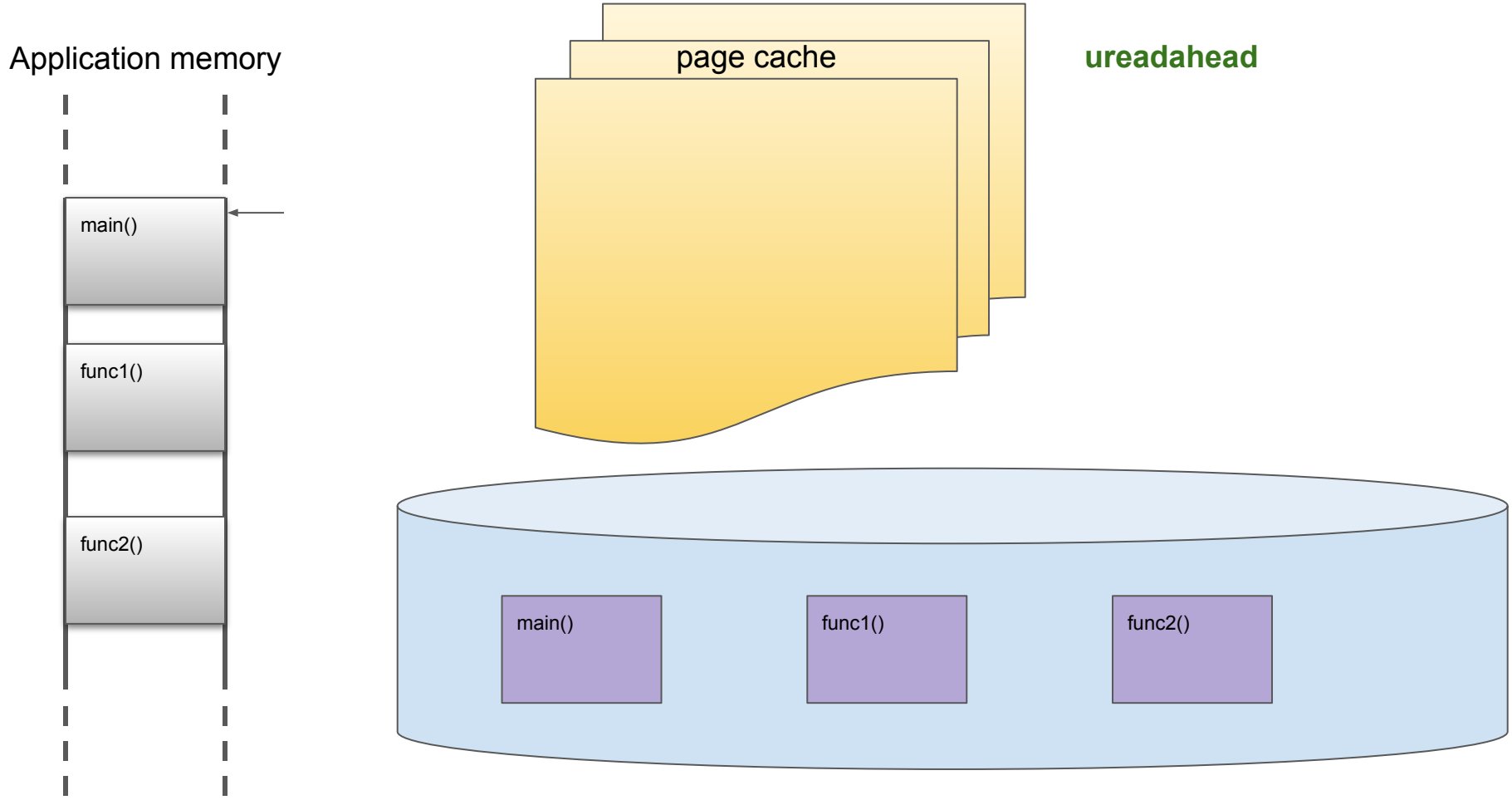
- The first boot traces files opened
  - After the trace it reads the memory that is mapped
  - Creates a “pack” file
- The next boot reads the “pack” file
  - Calls `readahead()` system call to prefetch the data from disk

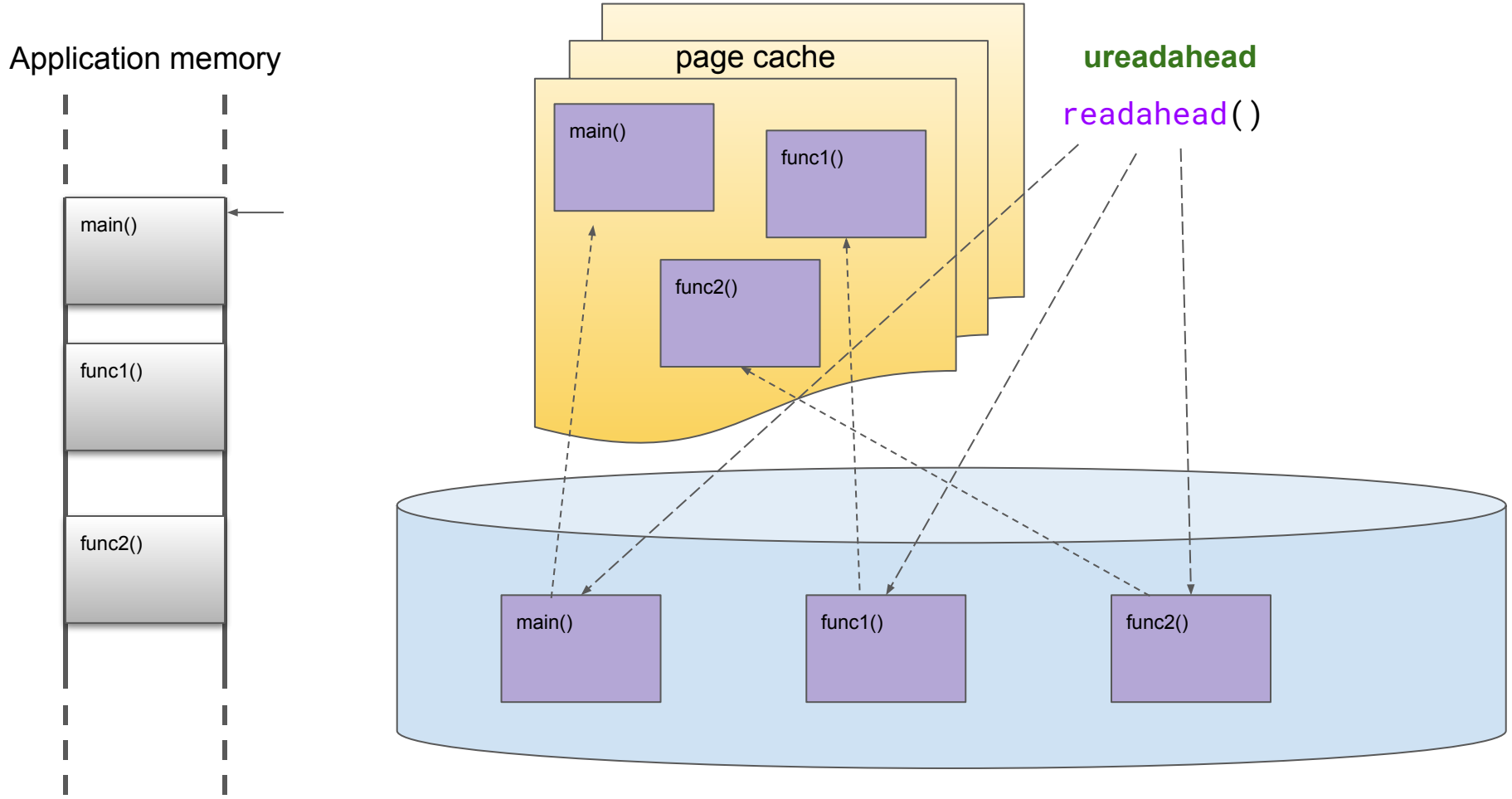
ureadahead records what is read

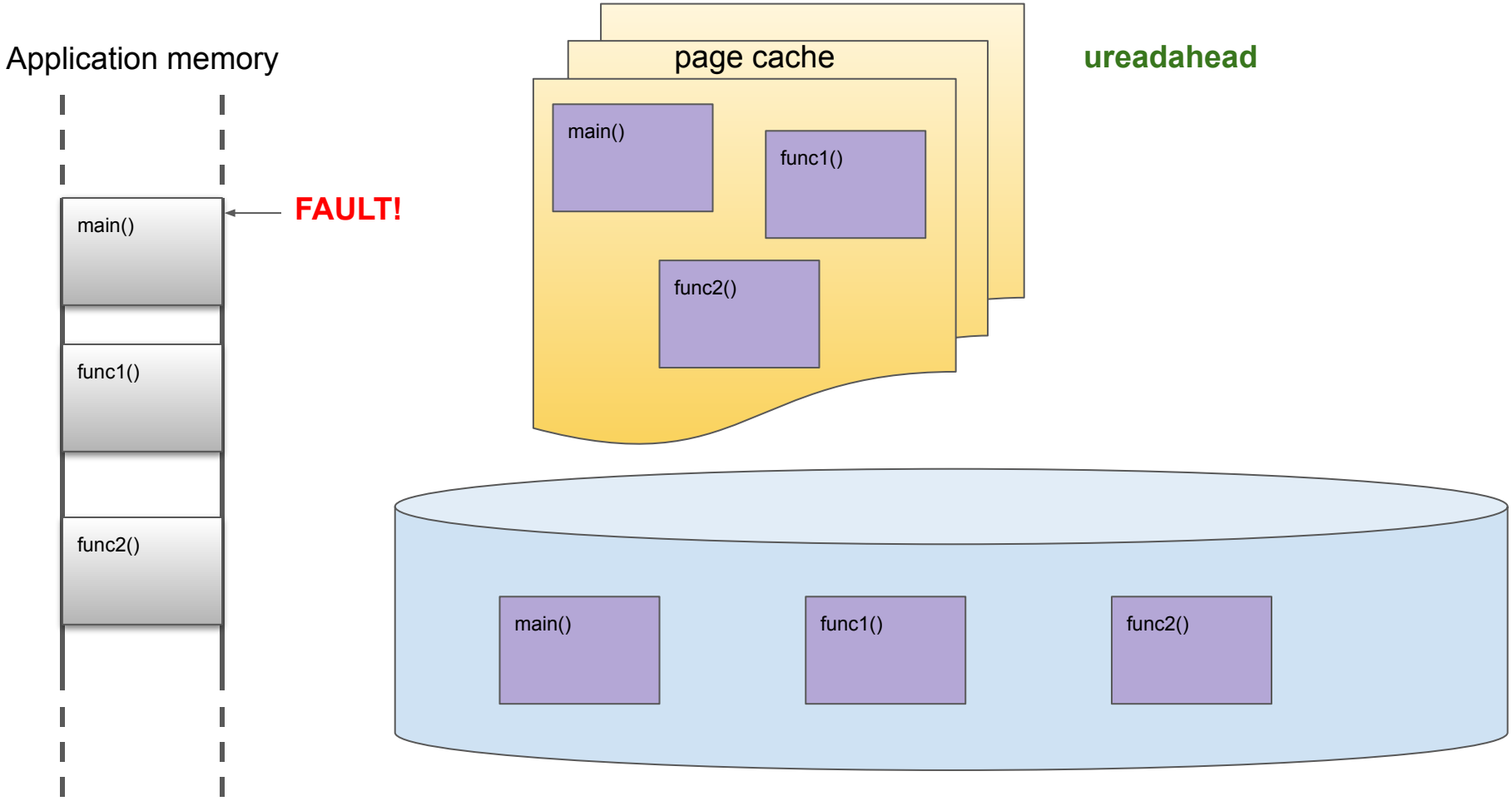
- The first boot traces files opened
  - After the trace it reads the memory that is mapped
  - Creates a “pack” file
- The next boot reads the “pack” file
  - Calls `readahead()` system call to prefetch the data from disk
  - Races with the application as they start

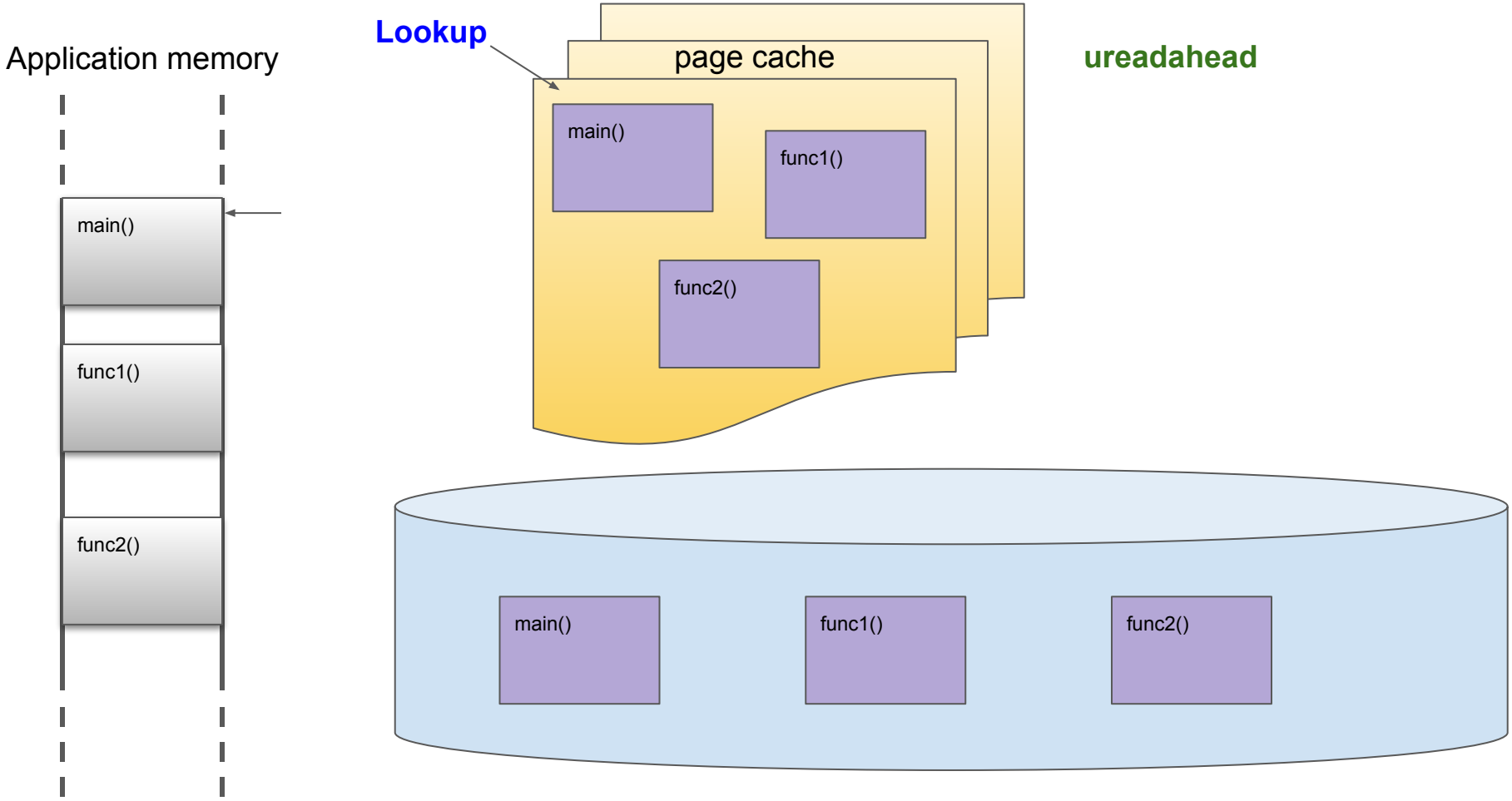
ureadahead records what is read

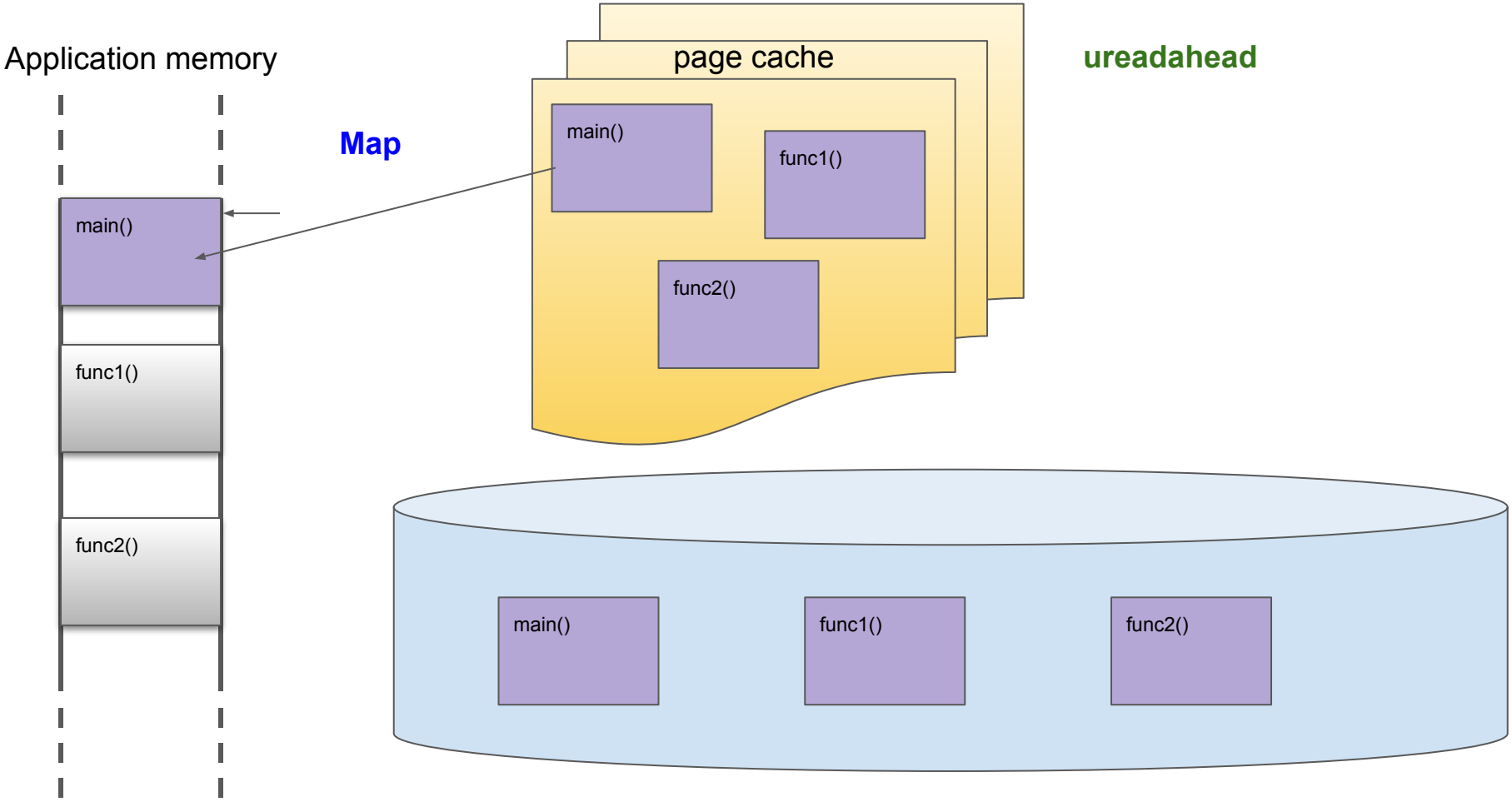
- The first boot traces files opened
  - After the trace it reads the memory that is mapped
  - Creates a “pack” file
- The next boot reads the “pack” file
  - Calls `readahead()` system call to prefetch the data from disk
  - Races with the application as they start
  - But still has good results


Application memory
main()
func1()
func2()
page cache
ureadahead
main()
func1()
func2()


Application memory
main()
func1()
func2()
page cache
main()
func1()
func2()
ureadahead
readahead()
main()
func1()
func2()


Application memory
main()
FAULT!
func1()
func2()
page cache
main()
func1()
func2()
ureadahead
main()
func1()
func2()


Application memory
main()
func1()
func2()
Lookup
page cache
main()
func1()
func2()
ureadahead
main()
func1()
func2()


Application memory
Map
page cache
ureadahead
main()
func1()
func2()
main()
func1()
func2()
main()
func1()
func2()

```
>#  ureadahead --dump

[..]
/usr/lib/x86_64-linux-gnu/libdb-5.3.so (1800 kB), 4 blocks (396 kB)
  [.................@.................@########################################]
  [#######.....................................................................]
  [.....................................................................@######]
  [###############.....................................................@##########]
  [#######                                                                     ]

        65536, 4096 bytes (at 18446744073709551615)
        135168, 196608 bytes (at 18446744073709551615)
        1445888, 131072 bytes (at 18446744073709551615)
        1773568, 73728 bytes (at 18446744073709551615)
[..]
```

```
>#  ureadahead --dump

[..]
/usr/lib/x86_64-linux-gnu/libdb-5.3.so (1800 kB), 4 blocks (396 kB)
  [.................@.................@########################################]
  [#######.....................................................................]
  [.....................................................................@######]
  [###############.....................................................@##########]
  [#######                                                                     ]

        65536, 4096 bytes (at 18446744073709551615)
        135168, 196608 bytes (at 18446744073709551615)
        1445888, 131072 bytes (at 18446744073709551615)
        1773568, 73728 bytes (at 18446744073709551615)
[..]
```

offset

length

physical address
of block device

Why I care

- ChromeOS uses it

Why I care

- ChromeOS uses it
  - Ironically Scott James Remnant was not involved at all with it

Why I care

- ChromeOS uses it
  - Ironically Scott James Remnant was not involved at all with it
- ChromeOS testing showed significant improvements with it!

Why I care

- ChromeOS uses it
  - Ironically Scott James Remnant was not involved at all with it
- ChromeOS testing showed significant improvements with it!

```
$ bootperf -o /tmp/test-${BOARD} ${DUT}
```

Why I care

- ChromeOS uses it
  - Ironically Scott James Remnant was not involved at all with it
- ChromeOS testing showed significant improvements with it!

```
$ bootperf -o /tmp/test-${BOARD} ${DUT}
[..]
$ less /tmp/test-${BOARD}/run.001/summary/results.json
[..]
    "seconds_kernel_to_login": {
        "summary": {
            "units": "seconds",
            "improvement_direction": "down",
            "type": "list_of_scalar_values",
            "values": [
                7.445,
                6.275,
                6.642,
                6.175,
                6.261,
                6.118,
                6.648,
                6.642,
                6.241,
                6.273
            ]
        }
    },
```

Why I care

- ChromeOS uses it
  - Ironically Scott James Remnant was not involved at all with it
- ChromeOS testing showed significant improvements with it!

```
$ bootperf -o /tmp/test-${BOARD} ${DUT}
[..]
$ less /tmp/test-${BOARD}/run.001/summary/results.json
[..]
    "seconds_kernel_to_login": {
        "summary": {
            "units": "seconds",
            "improvement_direction": "down",
            "type": "list_of_scalar_values",
            "values": [
                7.445,
                6.275,
                6.642,
                6.175,
                6.261,
                6.118,
                6.648,
                6.642,
                6.241,
                6.273
            ]
        }
    },
```

First Boot (no pack file)

Why I care

- ChromeOS uses it
  - Ironically Scott James Remnant was not involved at all with it
- ChromeOS testing showed significant improvements with it!

```
$ bootperf -o /tmp/test-${BOARD} ${DUT}
[..]
$ less /tmp/test-${BOARD}/run.001/summary/results.json
[..]
    "seconds_kernel_to_login": {
        "summary": {
            "units": "seconds",
            "improvement_direction": "down",
            "type": "list_of_scalar_values",
            "values": [
                7.445,
                6.275,
                6.642,
                6.175,
                6.261,
                6.118,
                6.648,
                6.642,
                6.241,
                6.273
            ]
        }
    },
```

Subsequent Boots (with pack file)

Why I care

- ChromeOS uses it
  - Ironically Scott James Remnant was not involved at all with it
- ChromeOS testing showed significant improvements with it!

```
$ bootperf -o /tmp/test-${BOARD} ${DUT}
[..]
$ less /tmp/test-${BOARD}/run.001/summary/results.json
[..]
    "seconds_kernel_to_login": {
        "summary": {
            "units": "seconds",
            "improvement_direction": "down",
            "type": "list_of_scalar_values",
            "values": [
                7.445,
                6.275,
                6.642,
                6.175,
                6.261,
                6.118,
                6.648,
                6.642,
                6.241,
                6.273
            ]
        }
    },
```

Subsequent Boots (with pack file)

14.5% savings!

History of ureadahead

- Started in 2009 by Scott James Remnant at Canonical

History of ureadahead

- Started in 2009 by Scott James Remnant at Canonical
  - Again, he now works for Google!

History of ureadahead

- Started in 2009 by Scott James Remnant at Canonical
  - Again, he now works for Google!
- Adds two trace events to the kernel

History of ureadahead

- Started in 2009 by Scott James Remnant at Canonical
  - Again, he now works for Google!
- Adds two trace events to the kernel
  - do_sys_open
  - open_exec
  - uselib

History of ureadahead

- Started in 2009 by Scott James Remnant at Canonical
  - Again, he now works for Google!
- Adds two trace events to the kernel
  - do_sys_open
  - open_exec
  - uselib
    - Yes, I know that’s three, but the last one was used but not any more

History of ureadahead

- Started in 2009 by Scott James Remnant at Canonical
  - Again, he now works for Google!
- Adds two trace events to the kernel
  - do_sys_open
  - open_exec
  - uselib
    - Yes, I know that’s three, but the last one was used but not any more
  - Uses this information to find out what files were opened

History of ureadahead

- Started in 2009 by Scott James Remnant at Canonical
  - Again, he now works for Google!
- Adds two trace events to the kernel
  - do_sys_open
  - open_exec
  - uselib
    - Yes, I know that’s three, but the last one was used but not any more
  - Uses this information to find out what files were opened
  - But can not handle relative paths!

History of ureadahead

- Started in 2009 by Scott James Remnant at Canonical
  - Again, he now works for Google!
- Adds two trace events to the kernel
  - do_sys_open
  - open_exec
  - uselib
    - Yes, I know that’s three, but the last one was used but not any more
  - Uses this information to find out what files were opened
  - But can not handle relative paths!
- The trace events were NACK’d by the upstream maintainer

History of ureadahead

- Started in 2009 by Scott James Remnant at Canonical
  - Again, he now works for Google!
- Adds two trace events to the kernel
  - do_sys_open
  - open_exec
  - uselib
    - Yes, I know that’s three, but the last one was used but not any more
  - Uses this information to find out what files were opened
  - But can not handle relative paths!
- The trace events were NACK’d by the upstream maintainer
- Requires modification of the kernel to work

History of ureadahead

- Started in 2009 by Scott James Remnant at Canonical
  - Again, he now works for Google!
- Adds two trace events to the kernel
  - do_sys_open
  - open_exec
  - uselib
    - Yes, I know that's three, but the last one was used but not any more
  - Uses this information to find out what files were opened
  - But can not handle relative paths!
- The trace events were NACK'd by the upstream maintainer
- Requires modification of the kernel to work
- `mincore`() does not give any idea of what order the files are read

History of ureadahead

- In 2011 Scott James Remnant left Canonical

History of ureadahead

- In 2011 Scott James Remnant left Canonical for Google

History of ureadahead

- In 2011 Scott James Remnant left Canonical for Google
- ureadahead went into “maintenance mode”

History of ureadahead

- In 2011 Scott James Remnant left Canonical for Google
- ureadahead went into “maintenance mode”
  - This required forward porting the trace event patches

History of ureadahead

- In 2011 Scott James Remnant left Canonical for Google
- ureadahead went into “maintenance mode”
  - This required forward porting the trace event patches
- Nobody took over maintainership

History of ureadahead

- In 2011 Scott James Remnant left Canonical for Google
- ureadahead went into “maintenance mode”
  - This required forward porting the trace event patches
- Nobody took over maintainership
- Now unsupported by Canonical

History of ureadahead

- In 2011 Scott James Remnant left Canonical for Google
- ureadahead went into “maintenance mode”
  - This required forward porting the trace event patches
- Nobody took over maintainership
- Now unsupported by Canonical
- The trace event patches stopped being forward ported by Canonical

History of ureadahead

- In 2011 Scott James Remnant left Canonical for Google
- ureadahead went into “maintenance mode”
  - This required forward porting the trace event patches
- Nobody took over maintainership
- Now unsupported by Canonical
- The trace event patches stopped being forward ported by Canonical
  - ureadahead stopped working!

History of ureadahead

- In 2011 Scott James Remnant left Canonical for Google
- ureadahead went into “maintenance mode”
  - This required forward porting the trace event patches
- Nobody took over maintainership
- Now unsupported by Canonical
- The trace event patches stopped being forward ported by Canonical
  - ureadahead stopped working!
  - I guess nobody knew why

History of ureadahead

- In 2011 Scott James Remnant left Canonical for Google
- ureadahead went into “maintenance mode”
  - This required forward porting the trace event patches
- Nobody took over maintainership
- Now unsupported by Canonical
- The trace event patches stopped being forward ported by Canonical
  - ureadahead stopped working!
  - I guess nobody knew why
  - I guess they just thought it was broken

History of ureadahead

- In 2011 Scott James Remnant left Canonical for Google
- ureadahead went into “maintenance mode”
  - This required forward porting the trace event patches
- Nobody took over maintainership
- Now unsupported by Canonical
- The trace event patches stopped being forward ported by Canonical
  - ureadahead stopped working!
  - I guess nobody knew why
  - I guess they just thought it was broken
  - Canonical eventually stopped supporting it

History of ureadahead

- In 2011 Scott James Remnant left Canonical for Google
- ureadahead went into “maintenance mode”
  - This required forward porting the trace event patches
- Nobody took over maintainership
- Now unsupported by Canonical
- The trace event patches stopped being forward ported by Canonical
  - ureadahead stopped working!
  - I guess nobody knew why
  - I guess they just thought it was broken
  - Canonical eventually stopped supporting it
  - Last update was in 2017

ureadahead is dead; Long live ureadahead!

- ChromeOS is the last user of it
  - We maintain it and patch our kernel for the two needed trace events

ureadahead is dead; Long live ureadahead!

- ChromeOS is the last user of it
  - We maintain it and patch our kernel for the two needed trace events
  - No, Scott James Remnant does not help us with it.

ureadahead is dead; Long live ureadahead!

- ChromeOS is the last user of it
  - We maintain it and patch our kernel for the two needed trace events
  - No, Scott James Remnant does not help us with it.
- It is mostly held together with band-aid patches

ureadahead is dead; Long live ureadahead!

- ChromeOS is the last user of it
  - We maintain it and patch our kernel for the two needed trace events
  - No, Scott James Remnant does not help us with it.
- It is mostly held together with band-aid patches
- Breaks with certain updates to the kernel

ureadahead is dead; Long live ureadahead!

- ChromeOS is the last user of it
  - We maintain it and patch our kernel for the two needed trace events
  - No, Scott James Remnant does not help us with it.
- It is mostly held together with band-aid patches
- Breaks with certain updates to the kernel
- Needs a new rewrite

ureadahead is dead; Long live ureadahead!

- ChromeOS is the last user of it
  - We maintain it and patch our kernel for the two needed trace events
  - No, Scott James Remnant does not help us with it.
- It is mostly held together with band-aid patches
- Breaks with certain updates to the kernel
- Needs a new rewrite
- I decided to start doing so

ureadahead rewrite

- Use libtracefs
  - Interface to access the tracefs file system

ureadahead rewrite

- Use libtracefs
  - Interface to access the tracefs file system
  - Paths were hardcoded (tracefs is not guaranteed to be mounted at the default location)

ureadahead rewrite

- Use libtracefs
  - Interface to access the tracefs file system
  - Paths were hardcoded (tracefs is not guaranteed to be mounted at the default location)
  - libtracefs searches `/proc/mounts` to find it
  - libtracefs mounts it if not already mounted

ureadahead rewrite

- Use libtracefs
  - Interface to access the tracefs file system
  - Paths were hardcoded (tracefs is not guaranteed to be mounted at the default location)
  - libtracefs searches `/proc/mounts` to find it
  - libtracefs mounts it if not already mounted
- Remove use of the non mainline trace events

ureadahead rewrite

- Use libtracefs
  - Interface to access the tracefs file system
  - Paths were hardcoded (tracefs is not guaranteed to be mounted at the default location)
  - libtracefs searches `/proc/mounts` to find it
  - libtracefs mounts it if not already mounted
- Remove use of the non mainline trace events
- Tracing open calls can not handle relative paths

ureadahead rewrite

- Use libtracefs
  - Interface to access the tracefs file system
  - Paths were hardcoded (tracefs is not guaranteed to be mounted at the default location)
  - libtracefs searches `/proc/mounts` to find it
  - libtracefs mounts it if not already mounted
- Remove use of the non mainline trace events
- Tracing open calls can not handle relative paths
- Must be a better trace event to use

ureadahead rewrite

- Use libtracefs
  - Interface to access the tracefs file system
  - Paths were hardcoded (tracefs is not guaranteed to be mounted at the default location)
  - libtracefs searches `/proc/mounts` to find it
  - libtracefs mounts it if not already mounted
- Remove use of the non mainline trace events
- Tracing open calls can not handle relative paths
- Must be a better trace event to use
  - Remember that `mm_filemap_add_to_page_cache` event we used?

ureadahead rewrite

- Use libtracefs
  - Interface to access the tracefs file system
  - Paths were hardcoded (tracefs is not guaranteed to be mounted at the default location)
  - libtracefs searches `/proc/mounts` to find it
  - libtracefs mounts it if not already mounted
- Remove use of the non mainline trace events
- Tracing open calls can not handle relative paths
- Must be a better trace event to use
  - Remember that `mm_filemap_add_to_page_cache` event we used?
- Can trace even the order pages were mapped in

ureadahead rewrite

- Use libtracefs
  - Interface to access the tracefs file system
  - Paths were hardcoded (tracefs is not guaranteed to be mounted at the default location)
  - libtracefs searches `/proc/mounts` to find it
  - libtracefs mounts it if not already mounted
- Remove use of the non mainline trace events
- Tracing open calls can not handle relative paths
- Must be a better trace event to use
  - Remember that `mm_filemap_add_to_page_cache` event we used?
- Can trace even the order pages were mapped in
- Doesn’t even care about “relative paths”

ureadahead rewrite

- Use the `mm_filemap_add_to_page_cache` event

```
chrome-4098  [004] 154614.283083: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x2711c3 ofs=1196032 order=0
```

ureadahead rewrite

- Use the `mm_filemap_add_to_page_cache` event

```
chrome-4098  [004] 154614.283083: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x2711c3 ofs=1196032 order=0
```

- Check `/proc/self/mountinfo`

```
28 1 254:3 / / rw,relatime shared:1 - ext4 /dev/vda3 rw,errors=remount-ro
```

ureadahead rewrite

- Use the `mm_filemap_add_to_page_cache` event

```
chrome-4098  [004] 154614.283083: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x2711c3 ofs=1196032 order=0
```

- Check `/proc/self/mountinfo`

```
28 1 254:3 / / rw,relatime shared:1 - ext4 /dev/vda3 rw,errors=remount-ro
```

- Searches the files on the device for a matching inode number
  - Uses `getdents64`() to quickly find files
  - Returns several inodes at once with the file names attached

ureadahead rewrite

- Use the `mm_filemap_add_to_page_cache` event

```
chrome-4098  [004] 154614.283083: mm_filemap_add_to_page_cache: dev 254:3 ino 13f82f pfn=0x2711c3 ofs=1196032 order=0
```

- Check `/proc/self/mountinfo`

```
28 1 254:3 / / rw,relatime shared:1 - ext4 /dev/vda3 rw,errors=remount-ro
```

- Searches the files on the device for a matching inode number
  - Uses `getdents64`() to quickly find files
  - Returns several inodes at once with the file names attached

https://github.com/rostedt/ureadahead/tree/devel

Much more to do!

Much more to do!

- Split the tracing and creation of the pack file from reading it

Much more to do!

- Split the tracing and creation of the pack file from reading it
  - One application to just trace and create the file

Much more to do!

- Split the tracing and creation of the pack file from reading it
  - One application to just trace and create the file
  - One application that reads the pack file and calls `readahead`()

Much more to do!

- Split the tracing and creation of the pack file from reading it
  - One application to just trace and create the file
  - One application that reads the pack file and calls `readahead`()
  - Make a series of pack files for different use cases

Much more to do!

- Split the tracing and creation of the pack file from reading it
  - One application to just trace and create the file
  - One application that reads the pack file and calls `readahead`()
  - Make a series of pack files for different use cases
- Make it smarter

Much more to do!

- Split the tracing and creation of the pack file from reading it
  - One application to just trace and create the file
  - One application that reads the pack file and calls `readahead`()
  - Make a series of pack files for different use cases
- Make it smarter
  - Read the the portions of the file in order

Much more to do!

- Split the tracing and creation of the pack file from reading it
  - One application to just trace and create the file
  - One application that reads the pack file and calls `readahead`()
  - Make a series of pack files for different use cases
- Make it smarter
  - Read the the portions of the file in order
  - Know the timestamps

Much more to do!

- Split the tracing and creation of the pack file from reading it
  - One application to just trace and create the file
  - One application that reads the pack file and calls `readahead`()
  - Make a series of pack files for different use cases
- Make it smarter
  - Read the the portions of the file in order
  - Know the timestamps
    - Can skip things that are likely being read by the current application

Much more to do!

- Split the tracing and creation of the pack file from reading it
  - One application to just trace and create the file
  - One application that reads the pack file and calls `readahead`()
  - Make a series of pack files for different use cases
- Make it smarter
  - Read the the portions of the file in order
  - Know the timestamps
    - Can skip things that are likely being read by the current application
- Rewrite in Rust?

Much more to do!

- Split the tracing and creation of the pack file from reading it
  - One application to just trace and create the file
  - One application that reads the pack file and calls `readahead`()
  - Make a series of pack files for different use cases
- Make it smarter
  - Read the the portions of the file in order
  - Know the timestamps
    - Can skip things that are likely being read by the current application
- Rewrite in Rust?
- What other ideas do you have?

Questions?