TOSHIBA

Leading Innovation >>>

Research and evaluation of SCHED_DEADLINE

Masahiro Yamada
Advanced Software Technology Group
Corporate Software Engineering Center
TOSHIBA CORPORATION

Copyright 2012, Toshiba Corporation.

Outline

- **Motivation**
- **SCHED_DEADLINEについて**
- **評価**
- **Conclusion**

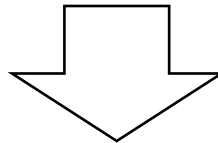
Outline

- **Motivation**
- SCHED_DEADLINEについて
- 評価
- Conclusion

Motivation

- 背景
  - 社会インフラ製品にLinux適用事例が増加
    - 豊富なドライバやライブラリへの要求，低コスト化
  - リアルタイム性の確保が重要に.

  ⇒現状は，RTパッチ＋SCHED_FIFOなどで対応.
- 問題点
  - 確実なデッドライン保障を行えるかどうかは，検証が必要.
  - 優先度の高い処理に不具合があった場合，ほかの処理に影響が及ぶ.



デッドライン保障を行うスケジューラが必要

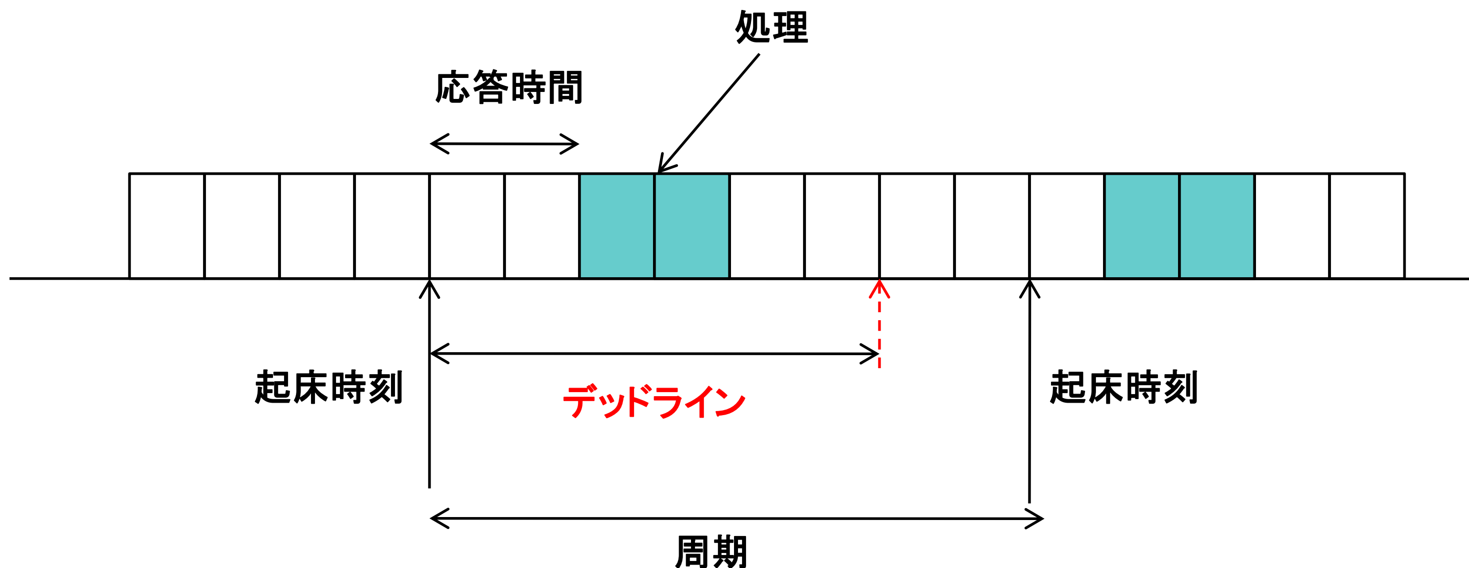
Eariest Deadline Fisrt scheduling（EDF）の特徴

- **デッドラインが最も近いタスクが最優先.**
- **タスクの優先度は動的に変化. SCHED_FIFOなどは静的優先度**
- **理論上CPU利用率を100%まで利用可能**
  - 実際はカーネルが動作する分のオーバヘッドを考慮する必要あり.
  - CPU利用100%以内ならばデッドライン保障が可能⇒検証が不要

タスクモデル

- 起床時刻＝タイマ割り込みなどのイベントによってタスクが実行可能
- 応答時間＝イベント発生から実際に処理を開始するまでの時間
- デッドライン＝起床時刻から処理を完了しなくてはならない時間

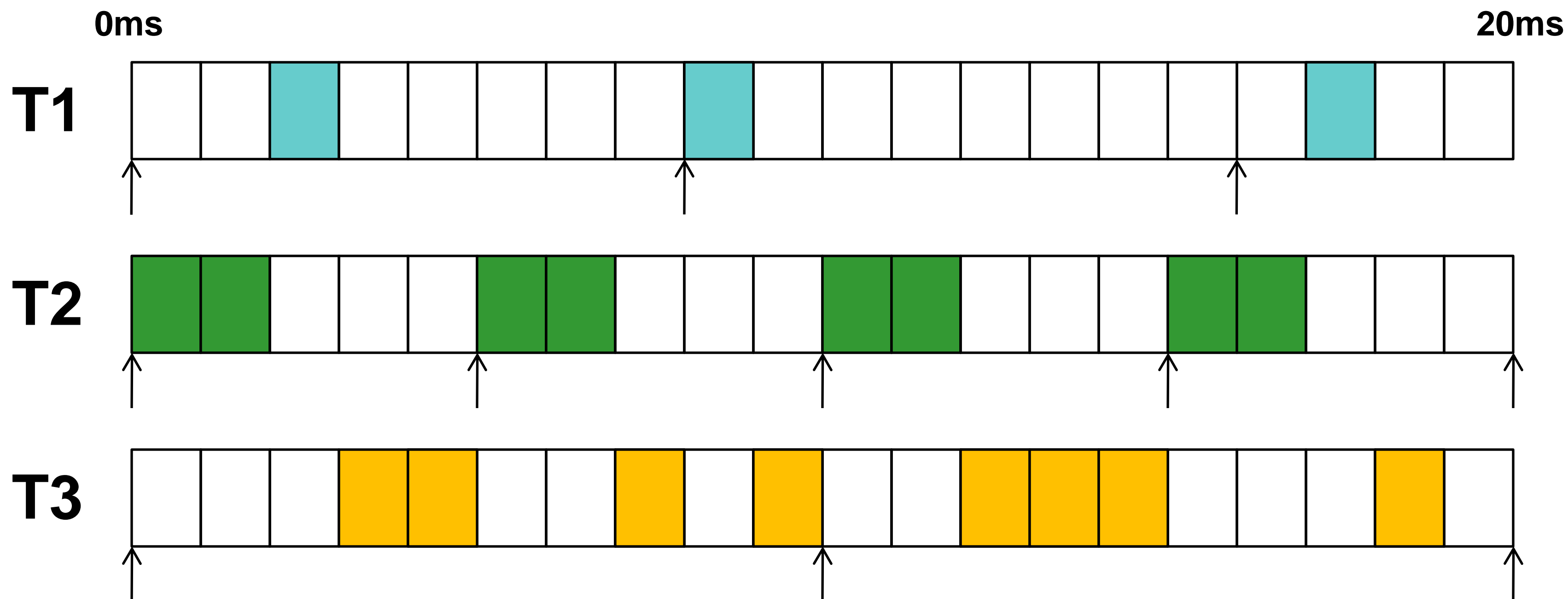
※以後，話を単純にするため デッドライン＝周期 の前提で話を進めます．



The Example of EDF Scheduling

- Task1: runtime 1ms period 8ms
- Task2: runtime 2ms period 5ms
- Task3: runtime 4ms period 10ms

CPU usage＝0.925% ＜ 100%



Rate-Monotonic Scheduling（RMS）

- **RTOSでよく利用されるスケジューリングアルゴリズム**
- **タスク(プロセス)の前提**
  - リソース(HW,キュー，セマフォ)を共有しない
  - デッドラインが既知で周期と一致
  - 固定優先度で周期が短いタスクに高い優先度を与える
- **CPU利用率の上限**
  - n:タスク数，Ti=タスクiの周期，Ci=タスクiの最悪実行時間

$$U = \sum_{i=0}^{n} C_i / T_i \le n(\sqrt[n]{2} - 1) \xrightarrow[n=\infty]{} \ln 2 \approx 0.69$$

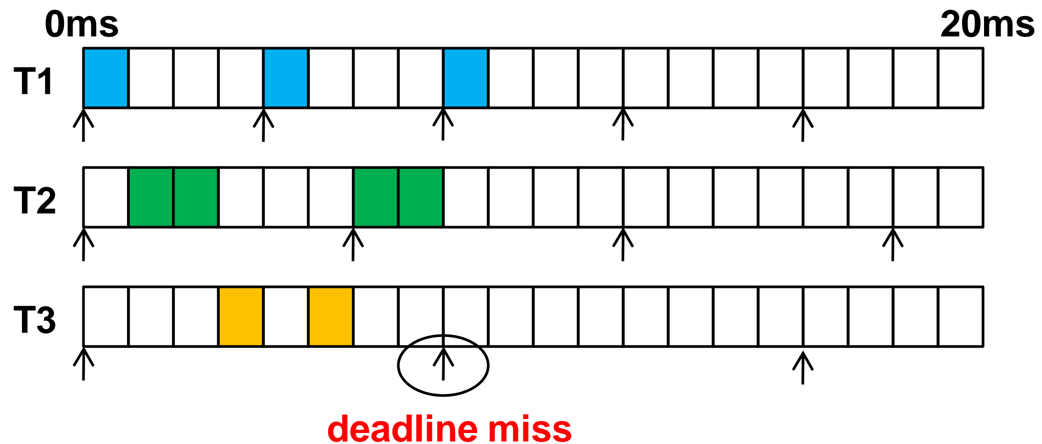
  - 実際はタスクの組み合わせに依存しており，80%前後のCPU利用率ならデッドラインミスが起きるタスクの組み合わせの可能性はまれ

RMSとEDFの比較

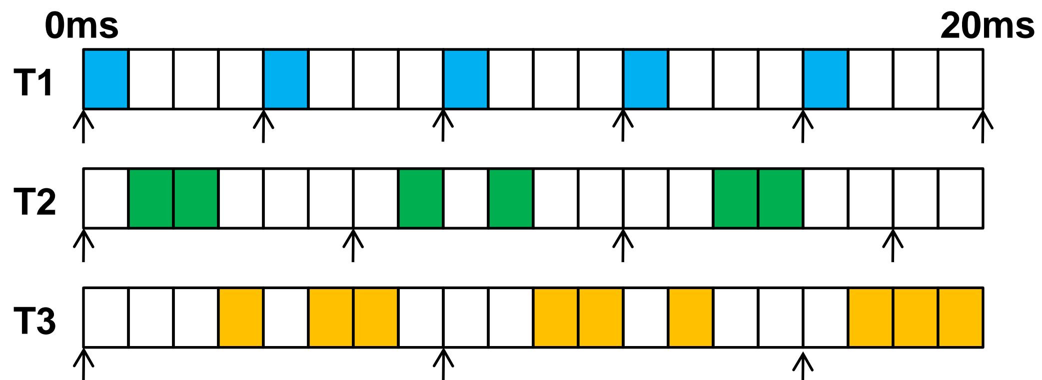
- Task1: runtime 1ms period 4ms
- Task2: runtime 2ms period 6ms
- Task3: runtime 3ms period 8ms

CPU usage＝0.958%

RMSで行う場合



EDFで行う場合



RMSとEDFの比較

	メリット	デメリット
RMS	スケジューラの実装が単純	タスクのスケジューリング可能性の検証が必要
EDF	タスクのスケジューリング可能性の検証は不要	スケジューラの実装が複雑

Outline

- Motivation
- **SCHED_DEADLINEについて**
- 評価
- Conclusion

SCHED_DEADLINEについて

- **デッドライン保障を行うLinuxカーネルのスケジューラ.**
- **Dario Faggioli氏が開発**
- **2011年12月現在，version3まで公開.**
  - ベースとなっているカーネルは2.6.33と2.6.36の2種類.
  - RTカーネル(2.6.33-rt)をベースにしたrt-deadlineもある.
- **EDF scheduling**
  - デッドラインの短い順にタスクを優先的に動作
- **temporal isolation**
  - CPUリソースをバジェットという単位で管理．バジェットがないタスクは次の周期に突入するまで実行不可.
  - 優先度の高いタスクに不具合があっても他のタスクの実行に影響をうけない

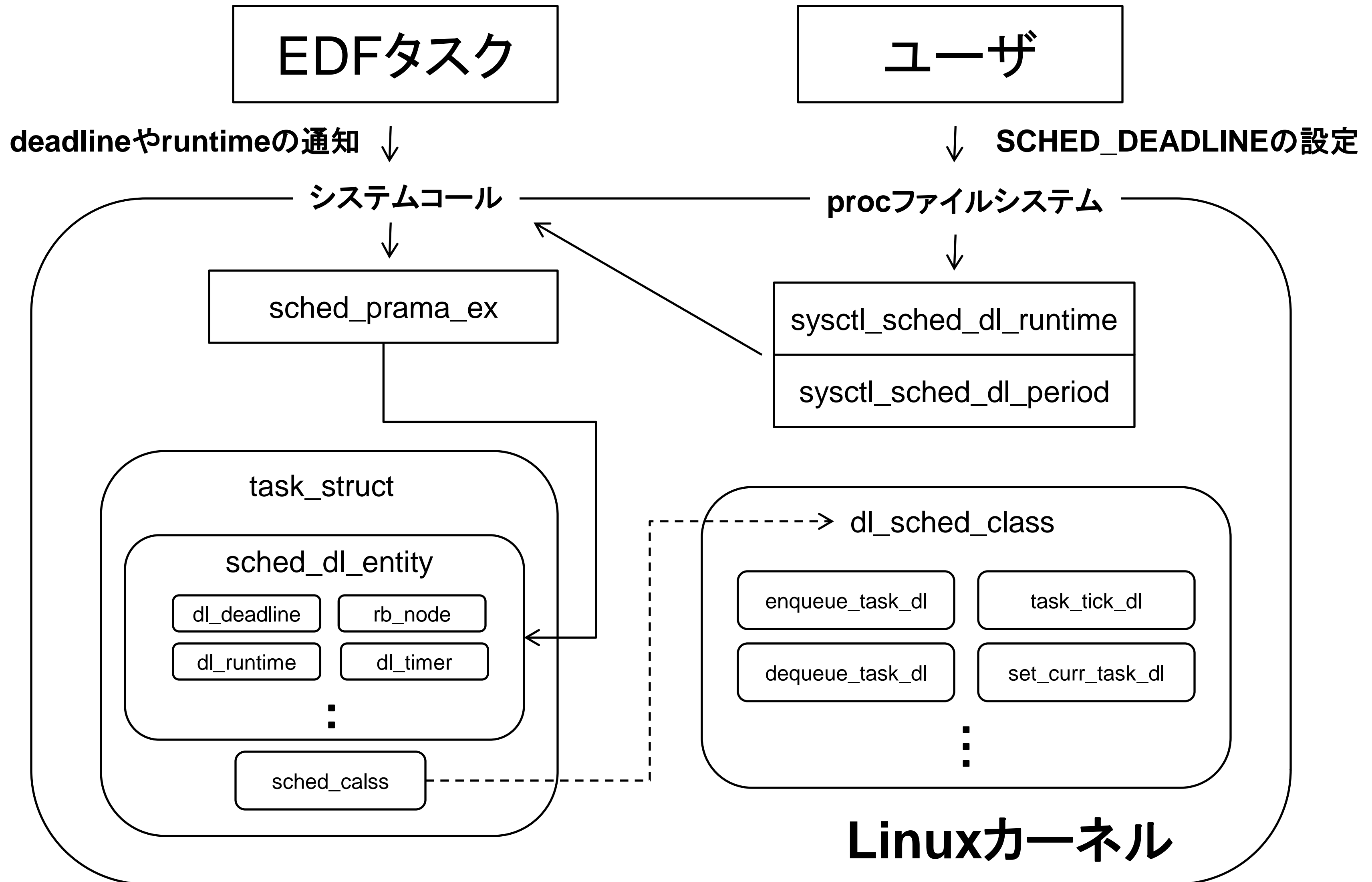
参考：http://www.evidence.eu.com/content/view/313/390/
http://gitorious.org/sched_deadline

Build SCHED_DEADLINE

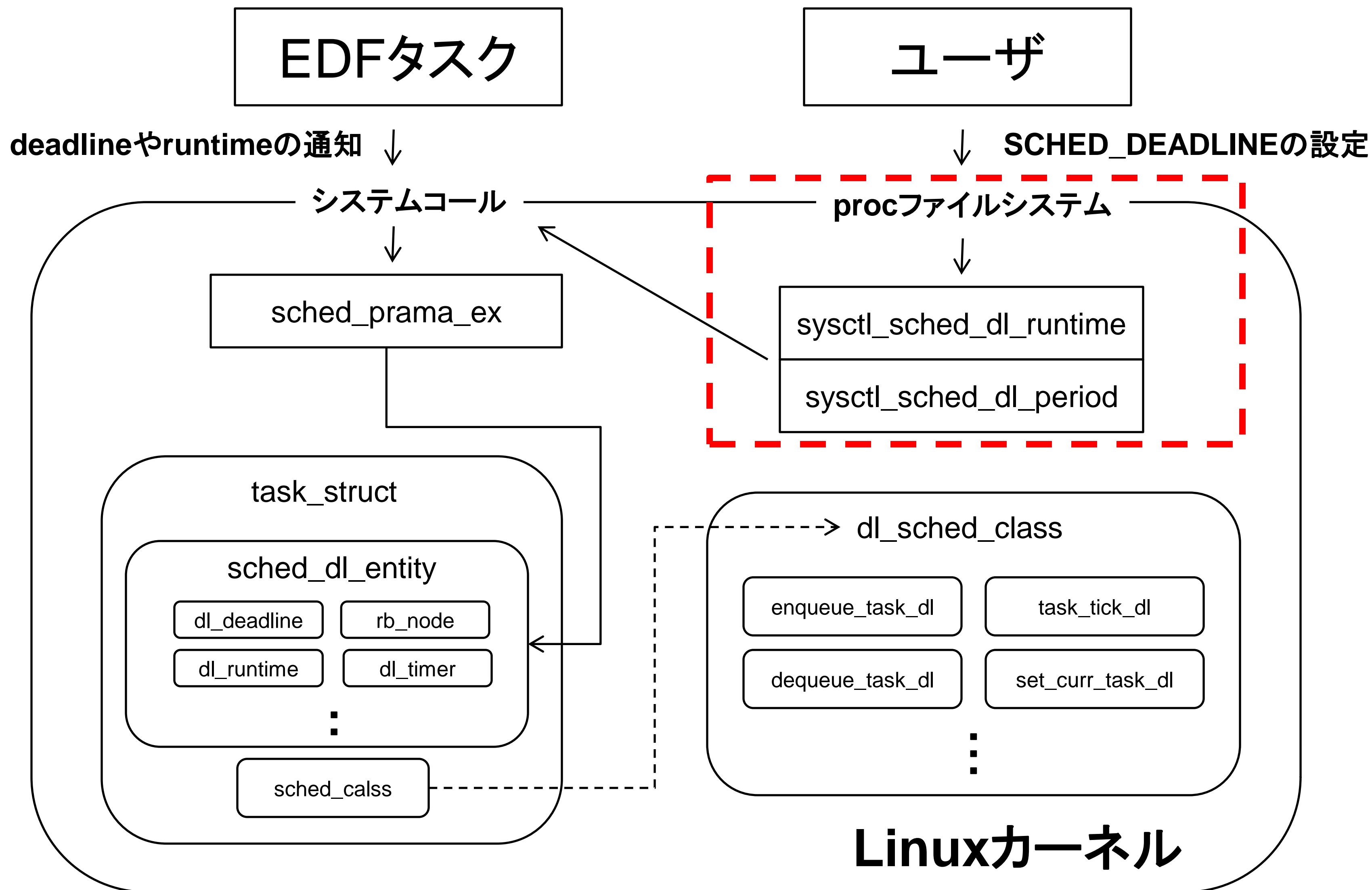
- **rt-deadlineの入手**
  - git clone git://gitorious.org/rt-deadline
  - v2を利用

- **Kernel configuration**
  - CONFIG_EXPERIMENTAL = y
  - CONFIG_CGROUPS = y
  - CONFIG_CGROUP_SCHED = n
  - CONFIG_HIGH_RES_TIMERS = y
  - CONFIG_PREEMPT = y
  - CONFIG_PREEMPT_RT = y

SCHED_DEADLINEの全体構成（概要）



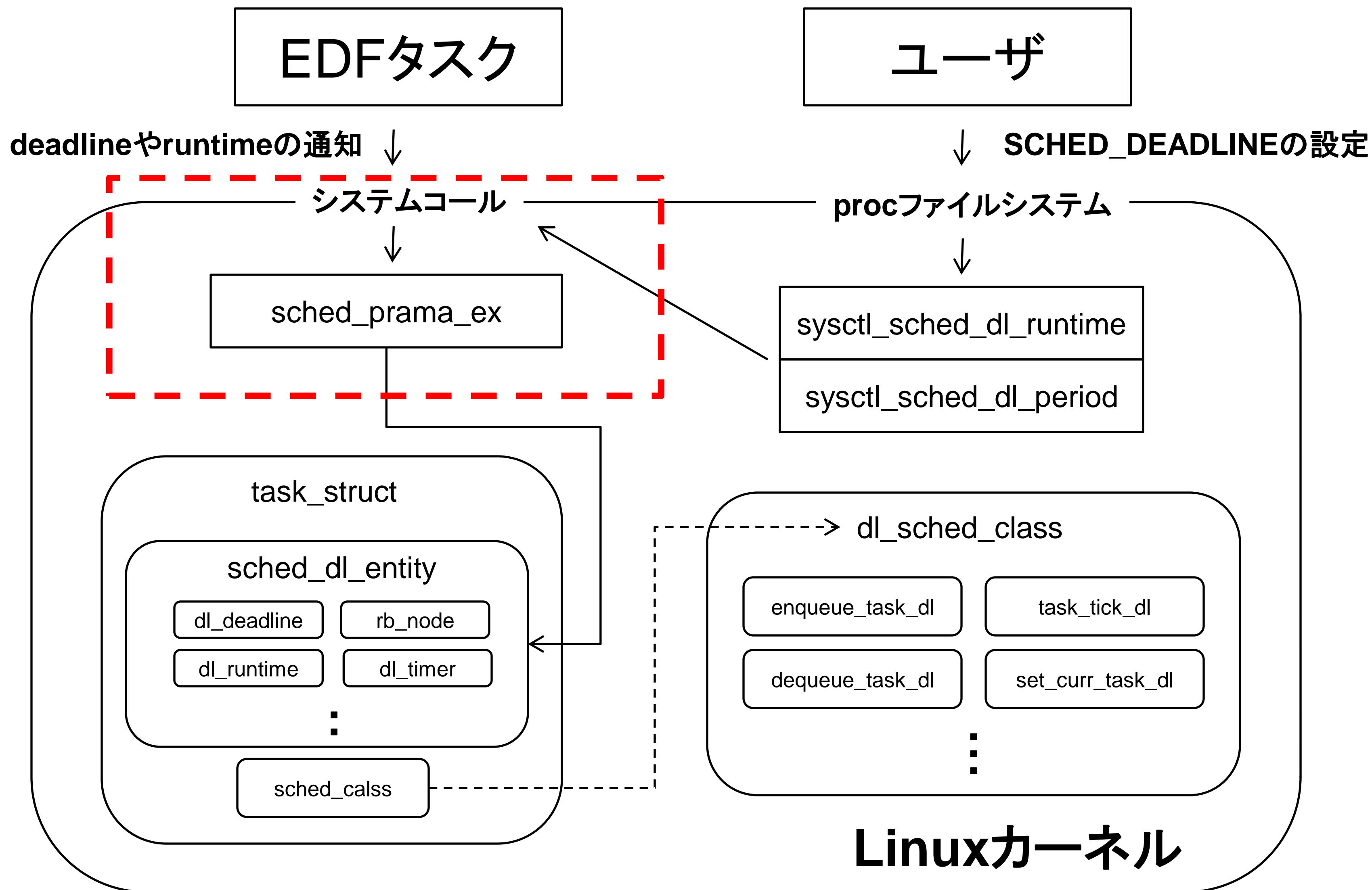
EDFスケジューラのCPUリソース設定



EDFスケジューラのCPUリソース設定

- **システム全体でどれくらいのCPUリソースをEDFスケジューラが管理するタスク全体に与えることができるかを設定しておく.**
- **procファイルシステムを用いて設定**
  - rt（SCHED_FIFOやSCHED_RR)＋dl(SCHED_DEADLINE)で100%.
  - EDFスケジューラに割り当てるCPUの周期と割合
    - /proc/sys/kernel/sched_dl_period_us
    - /proc/sys/kernel/sched_dl_runtime_us
  - ここで設定した値を超えるようなタスクはEDFスケジューラに登録できない
- **設定例（rtを50%，dlを50%）**
    - # echo 500000 > /proc/sys/kernel/sched_rt_runtime_us
    - # echo 100000 > /proc/sys/kernel/sched_dl_period_us
    - # echo 50000 > /proc/sys/kernel/sched_dl_runtime_us

EDF taskの実行

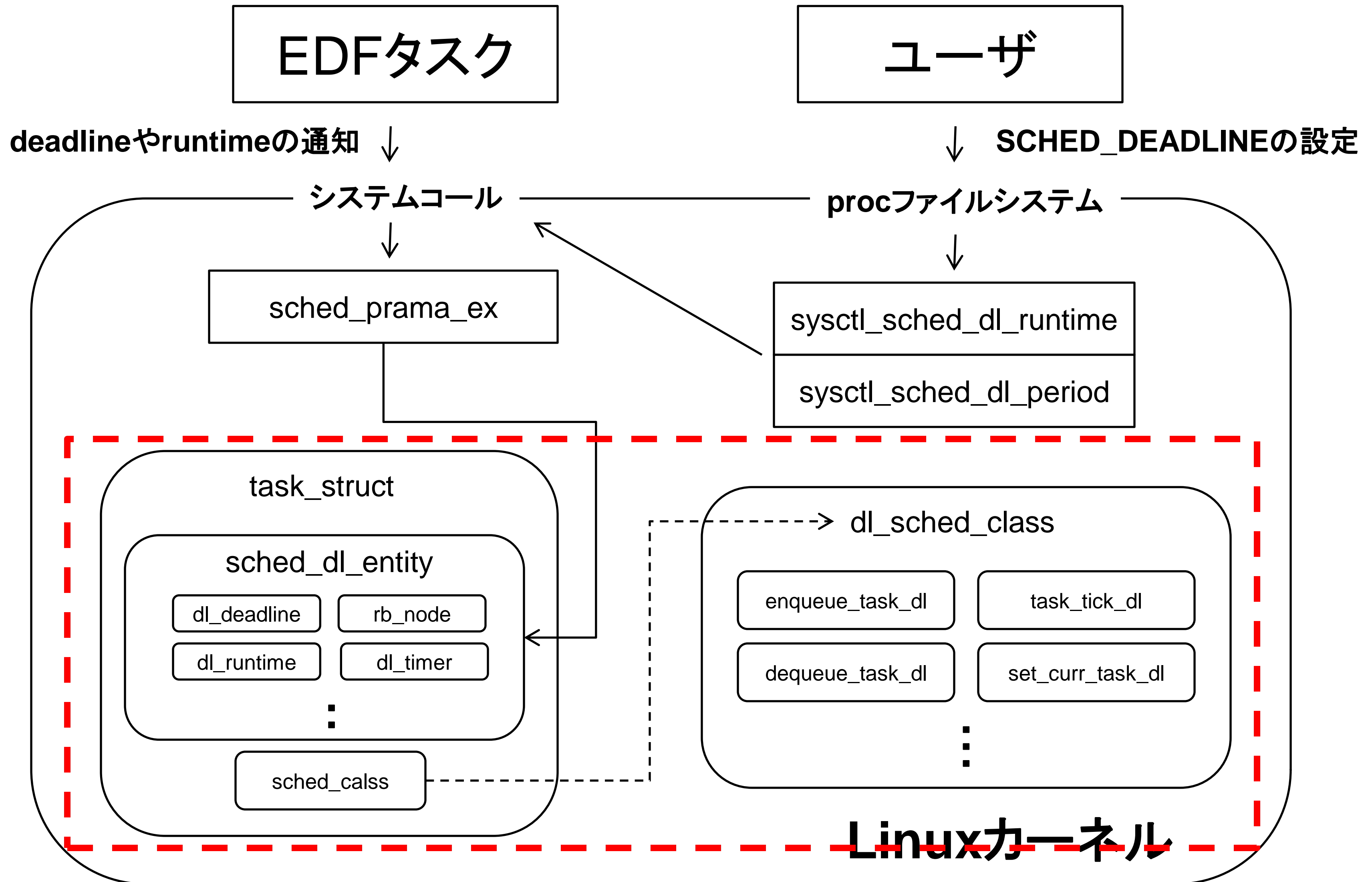


EDF taskの実行

- **schedtoolによるEDFタスクの実行**
  - # schedtool -E -t 10000:100000 -a 0 -e ./yes
  - オプション
    - -E : デッドラインタスク
    - -t : ＜実行時間(us) ＞ : ＜周期(us)＞
    - -a : 所属するCPUコアの指定
    - -e : 実行ファイルの指定

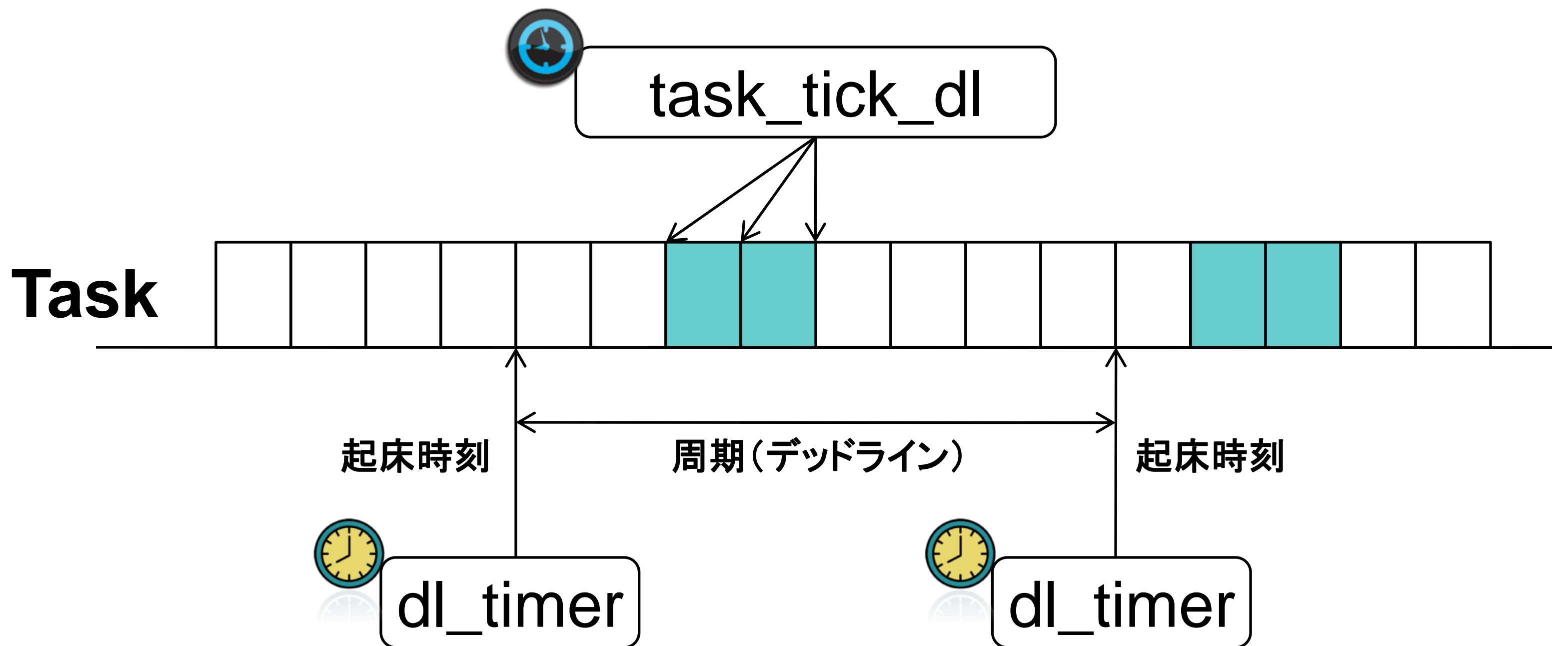
- **システムコールによるEDFタスクの実行**
  - タスクの実行開始後，周期や実行時間を設定して，以下のシステムコールに渡して実行
    - sched_setscheduler_ex

EDFタスクのバジェット管理



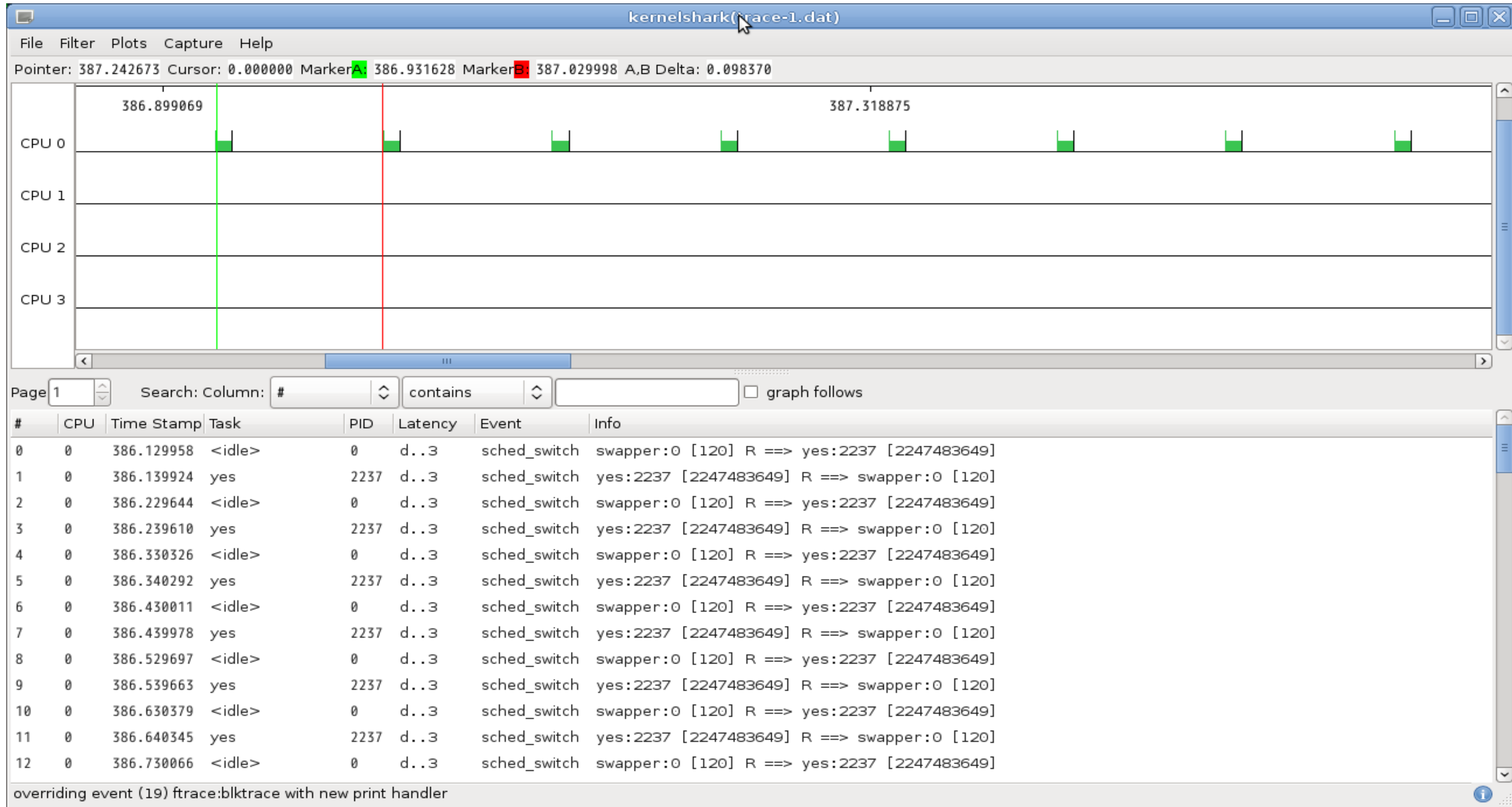
EDFタスクのバジェット管理

- SCHED_DEADLINEに登録されたタスクはCPUを占有できる時間(実行時間＝バジェット)を持つ.
- バジェット管理
  - バジェットの補充⇒ dl_timer (high resolution timer)
  - バジェットの消費⇒ task_tick_dl (tick依存)



1 EDF task

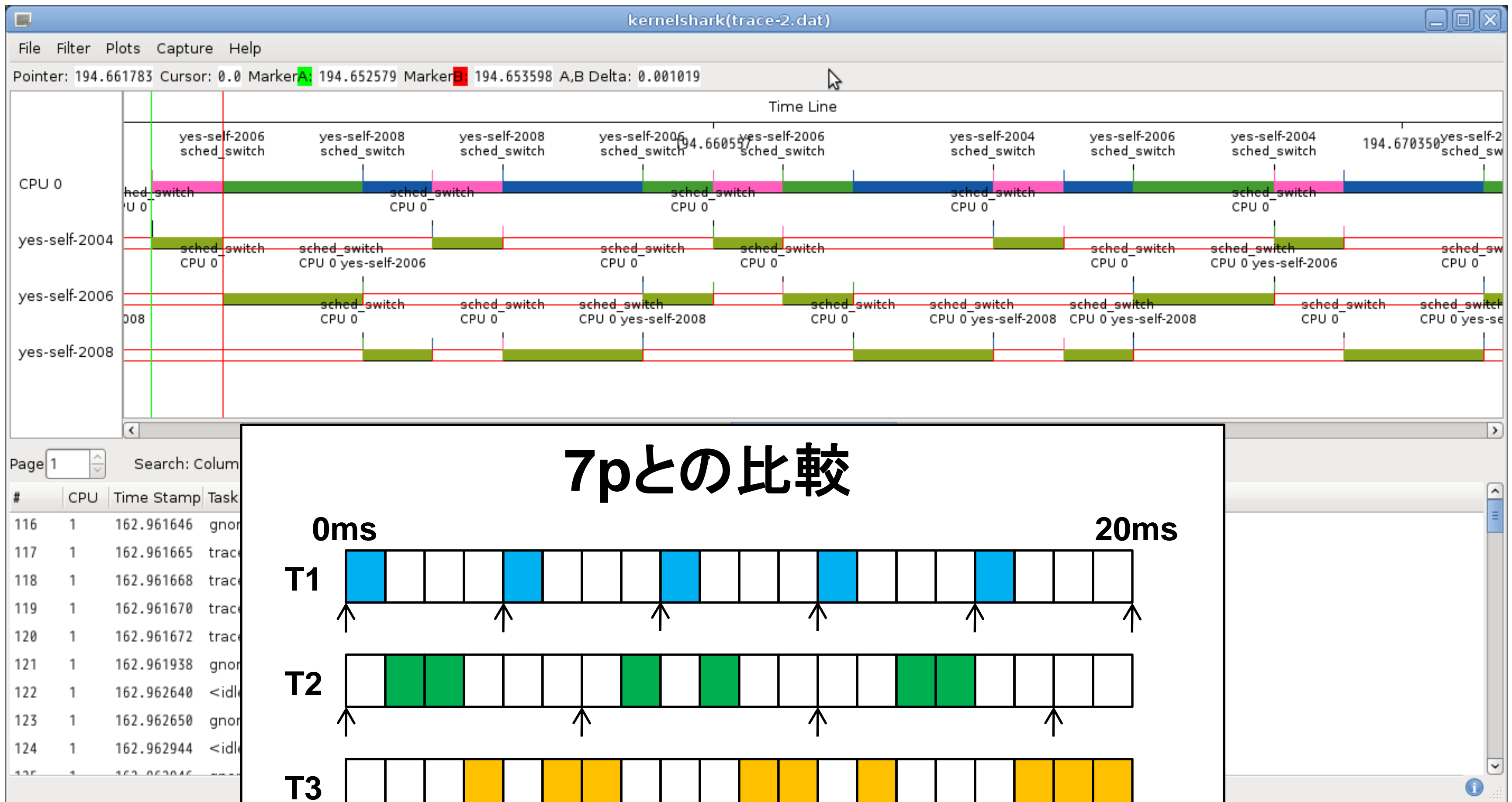
- **Task T1: runtime 10ms period 100ms**



#	CPU	Time Stamp	Task	PID	Latency	Event	Info
0	0	386.129958	<idle>	0	d..3	sched_switch	swapper:0 [120] R ==> yes:2237 [2247483649]
1	0	386.139924	yes	2237	d..3	sched_switch	yes:2237 [2247483649] R ==> swapper:0 [120]
2	0	386.229644	<idle>	0	d..3	sched_switch	swapper:0 [120] R ==> yes:2237 [2247483649]
3	0	386.239610	yes	2237	d..3	sched_switch	yes:2237 [2247483649] R ==> swapper:0 [120]
4	0	386.330326	<idle>	0	d..3	sched_switch	swapper:0 [120] R ==> yes:2237 [2247483649]
5	0	386.340292	yes	2237	d..3	sched_switch	yes:2237 [2247483649] R ==> swapper:0 [120]
6	0	386.430011	<idle>	0	d..3	sched_switch	swapper:0 [120] R ==> yes:2237 [2247483649]
7	0	386.439978	yes	2237	d..3	sched_switch	yes:2237 [2247483649] R ==> swapper:0 [120]
8	0	386.529697	<idle>	0	d..3	sched_switch	swapper:0 [120] R ==> yes:2237 [2247483649]
9	0	386.539663	yes	2237	d..3	sched_switch	yes:2237 [2247483649] R ==> swapper:0 [120]
10	0	386.630379	<idle>	0	d..3	sched_switch	swapper:0 [120] R ==> yes:2237 [2247483649]
11	0	386.640345	yes	2237	d..3	sched_switch	yes:2237 [2247483649] R ==> swapper:0 [120]
12	0	386.730066	<idle>	0	d..3	sched_switch	swapper:0 [120] R ==> yes:2237 [2247483649]

3 EDF tasks

- **Task T1: runtime 1ms period 4ms**
- **Task T2: runtime 2ms period 6ms**
- **Task T3: runtime 3ms period 8ms**



Outline

- Motivation
- SCHED_DEADLINEについて
- 評価
- Conclusion

評価

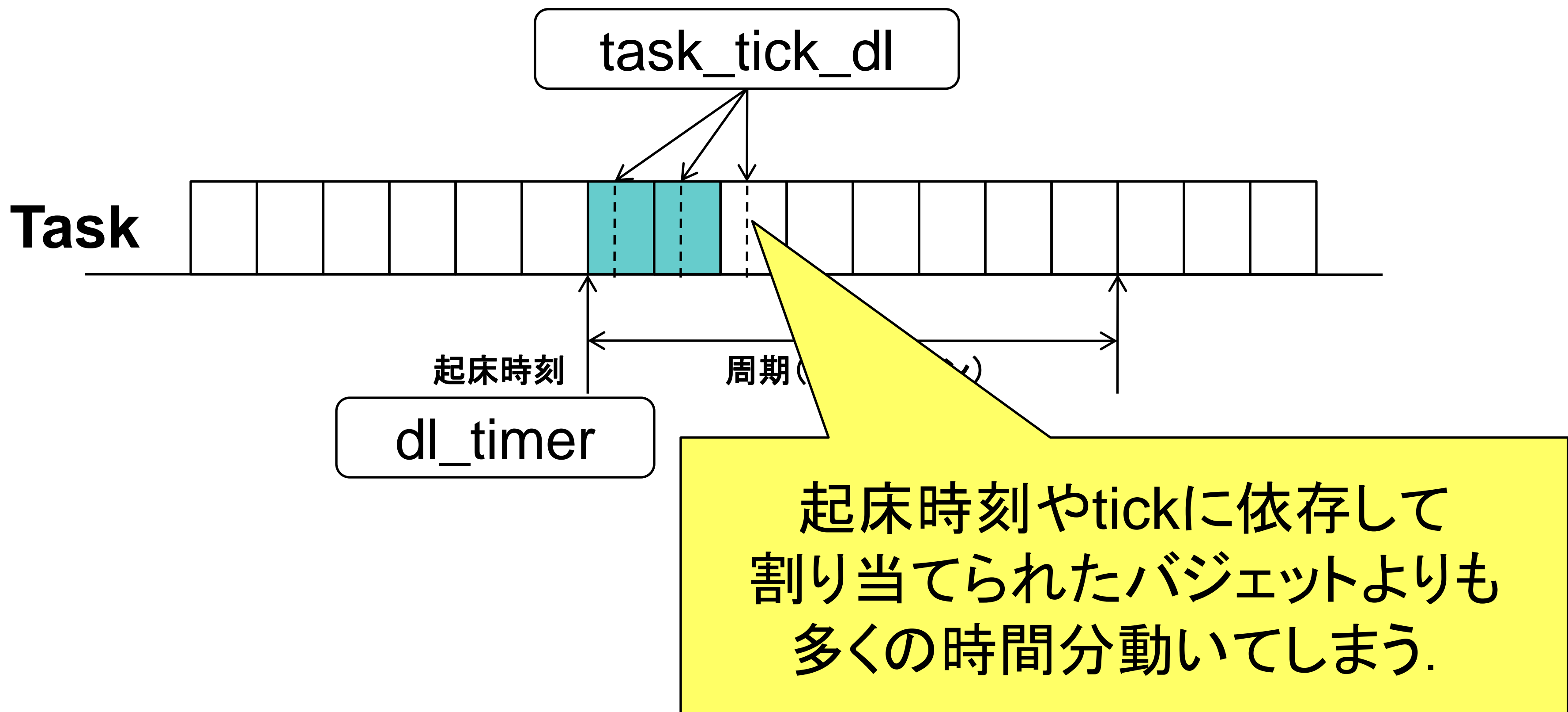
- 目的
  - システム全体として，どれくらいEDFタスクが動作することを許容できるか？
  - そもそもデッドラインが保障できているか？
- 方法
  - バジェットオーバランの測定
  - EDFタスクの周期・実行時間の限界値を調べる
  - trace-cmdにより実行時間の評価

動作させるEDFタスク

- 実行可能な状態の時は常にループ状態
- 周期(＝デッドライン)，実行時間は任意で設定できる
- バジェットオーバランの取得を行う.

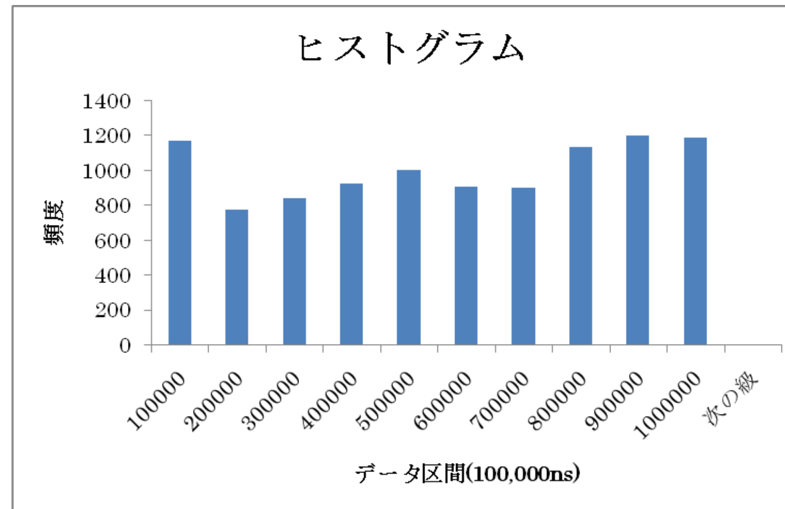
バジェットオーバラン測定

- **バジェットーオーバランの定義:**
  - 与えられたバジェットよりも多く処理をしてしまった時間
- **この値が大きいと，空きで使えるCPU使用率は減る.**



バジェットオーバラン測定

- CONFIG_HZ_1000 = y
  - <u>バジェットオーバラン最大値 < 1ms</u>
  - EDFタスクの周期やバジェットに依存せず.



- CONFIG_HZ_100 = y
  - <u>バジェットオーバラン最大値 < 10ms</u>

バジェットオーバラン測定

- CONFIG_HZ_1000 = y
  - バジェットオーバラン最大値 < 1ms
  - EDFタスクの周期やバジェットに依存せず.

ヒストグラム

HZに依存してオーバランの値が決定
バジェットオーバランが起きるのは
バジェットの補充と消費を行う処理の
時間粒度が異なるために発生していると考えられる

- CONFIG_HZ_100 = y
  - バジェットオーバラン最大値 < 10ms

EDFタスクの周期・実行時間の限界値

- **CONFIG_HZ_1000 = y**

usage

period

	40%	50%	60%	70%	80%	90%
30ms	○	○	○	○	○	○
20ms	○	○	○	○	○	○
10ms	○	○	○	○	○	×
5ms	○	○	○	○	○	×
4ms	○	○	○	○	×	×
3ms	○	○	○	×	×	×
2ms	○	×	×	×	×	×

○：動作可能
×：動作不能

EDFタスクの周期・実行時間の限界値

- **CONFIG_HZ_1000 = y**

usage

period

	40%	50%	60%	70%	80%	90%
30ms	○	○	○	○	○	○
20ms	○	○	○	○	○	○
10ms	○	○	○	○	○	×
5ms	○	○	○	○	○	×
4ms	○	○	○	○	×	×
3ms	○	○	○	×	×	×

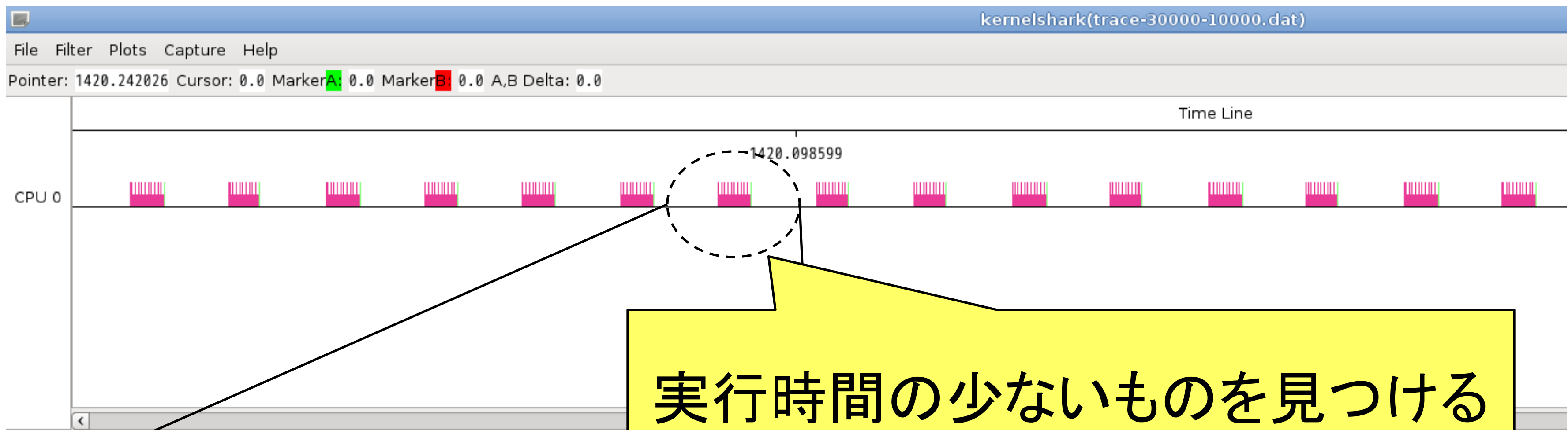
•CPUの空きを1ms以上確保しないとNG
•上の条件を満たしていない場合，いくらCPU利用率が低くてもシステムが動作しない場合がある．

trace-cmdにより実行時間の評価

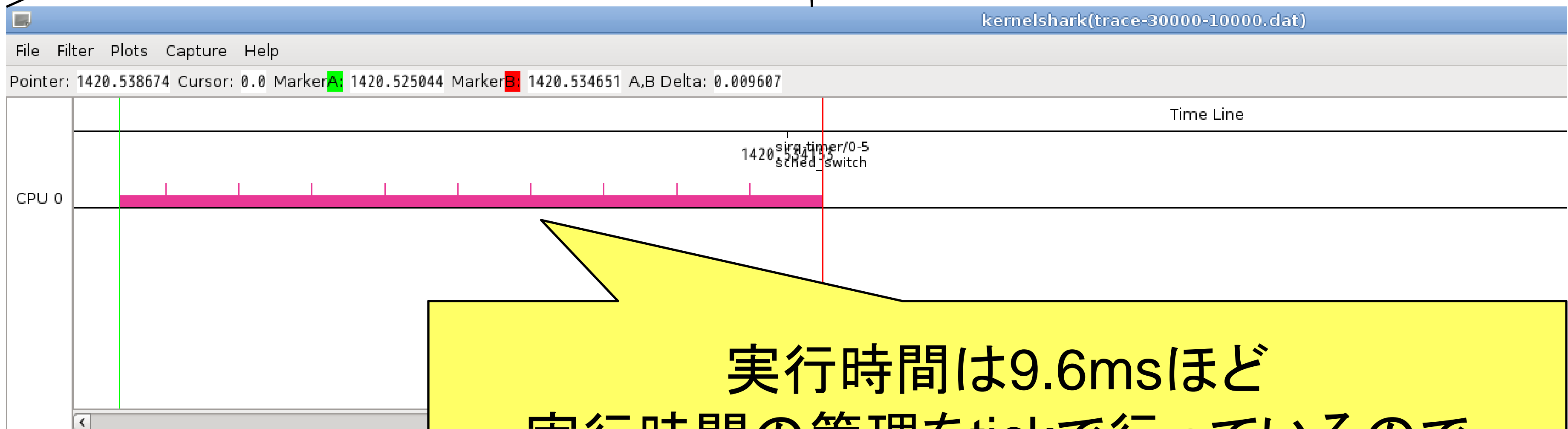
- **デッドライン保障の定義:**
  - 周期の間に必要な実行時間を与えられているかどうか？
- 設定
  - CONFIG_HZ_1000 = y
  - SCHED_DEADLINEに50%のCPU利用率を設定
- EDFタスク

周期	実行時間
30.0ms	10.0ms
3.0ms	1.0ms
30.5ms	10.0ms
30.0ms	10.5ms

周期30.0ms, 実行時間10.0ms

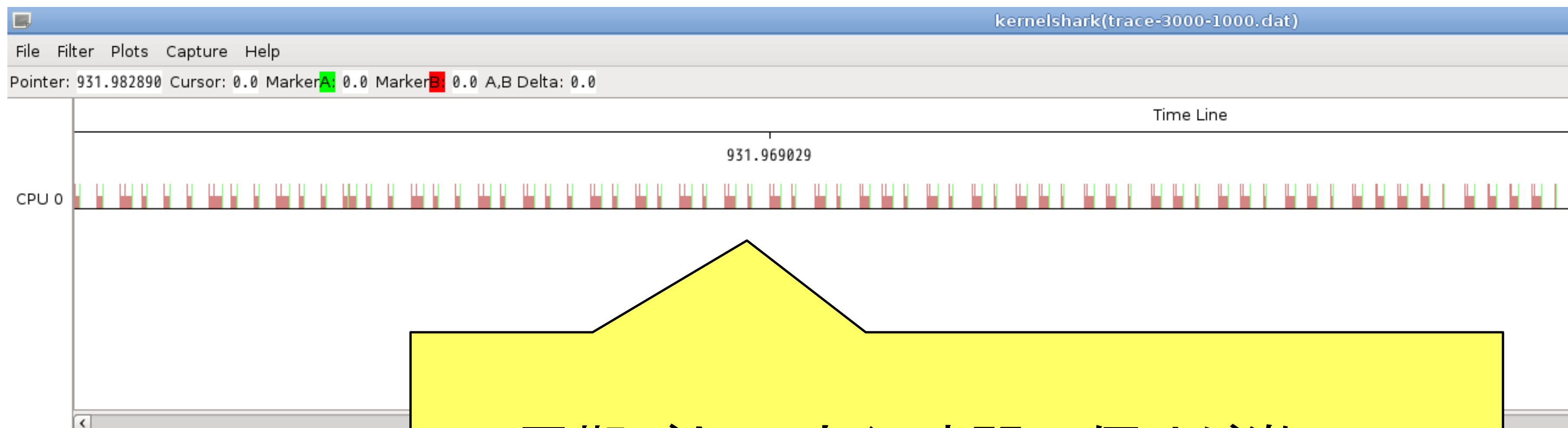


実行時間の少ないものを見つける

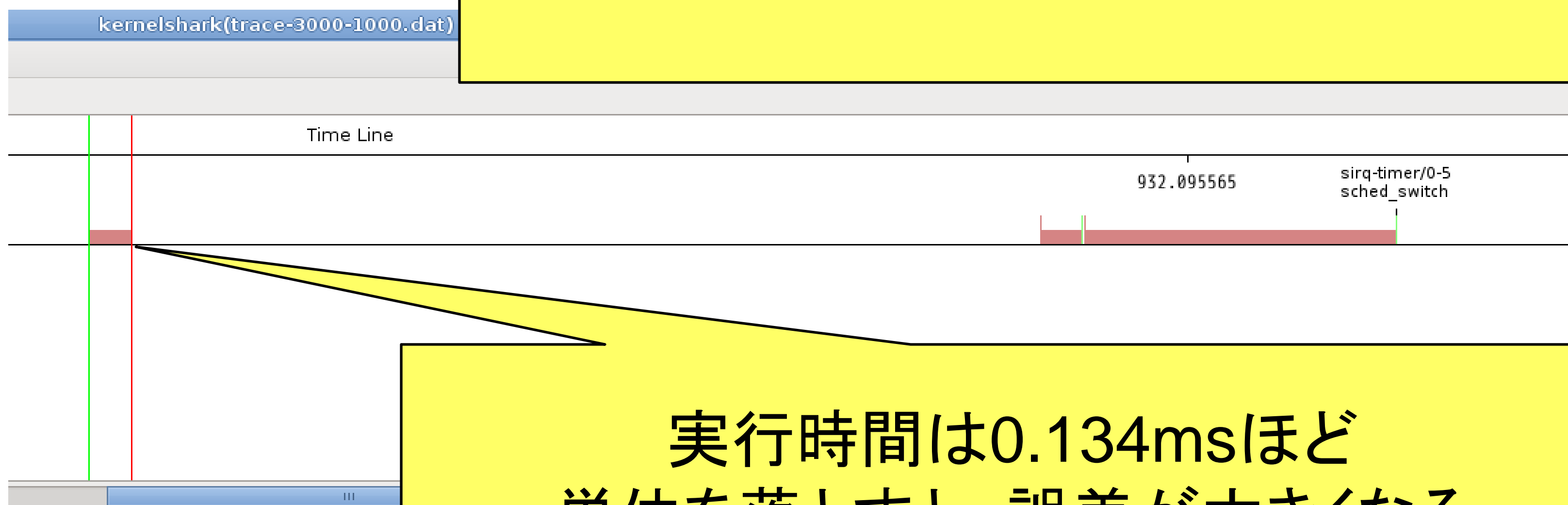


実行時間は9.6msほど
実行時間の管理をtickで行っているので
10回tickが発生するとOKと識別している

周期3.0ms，実行時間1.0ms

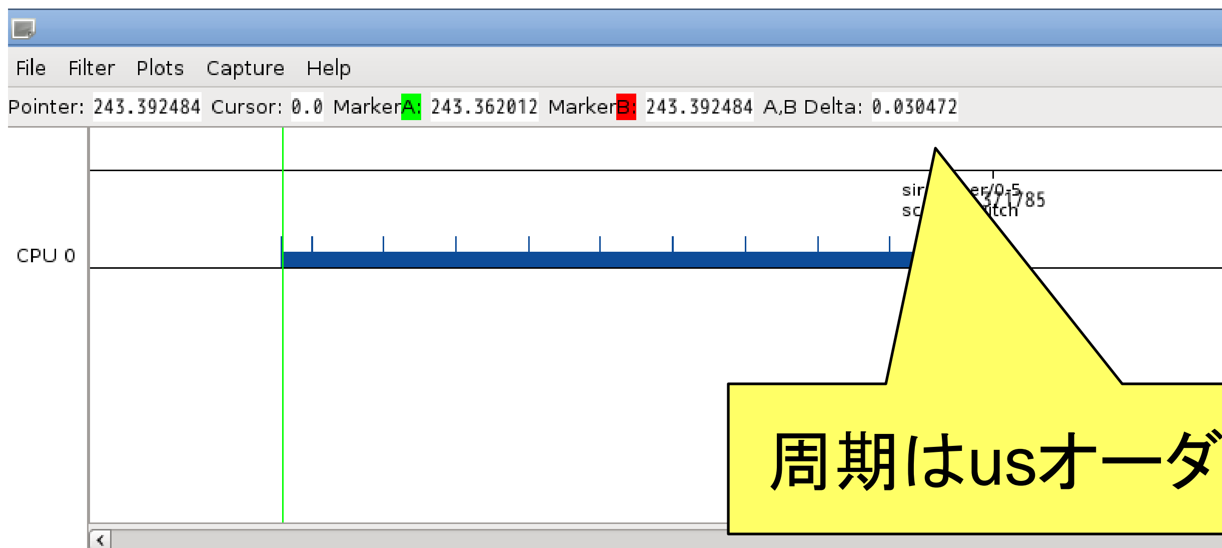


周期ごとに,実行時間の偏りが激しい

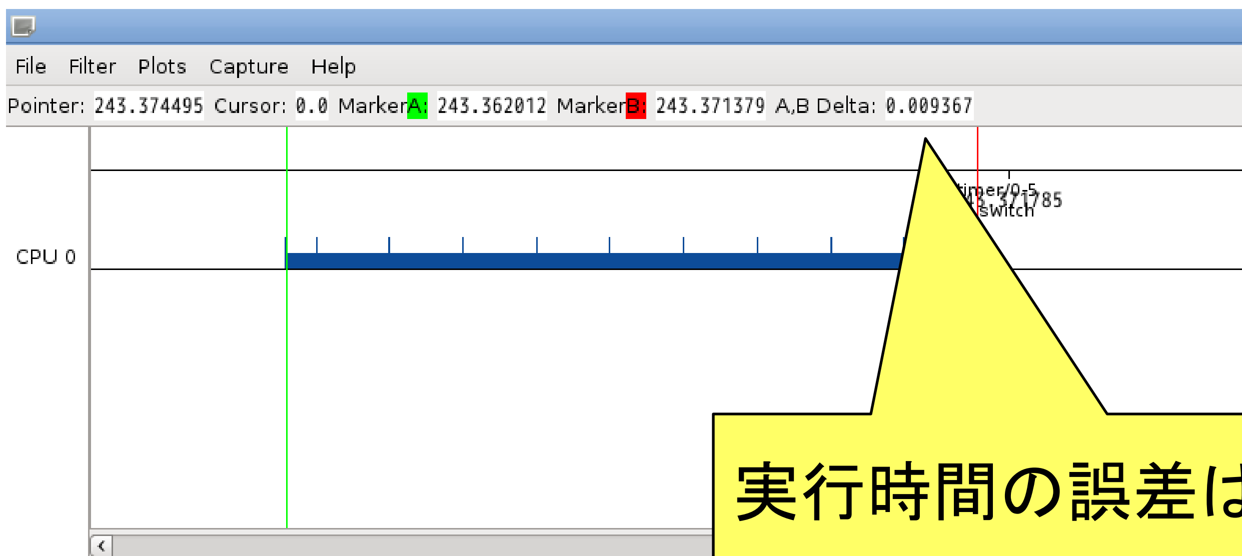


実行時間は0.134msほど
単位を落とすと，誤差が大きくなる

周期30.5ms，実行時間10.0ms

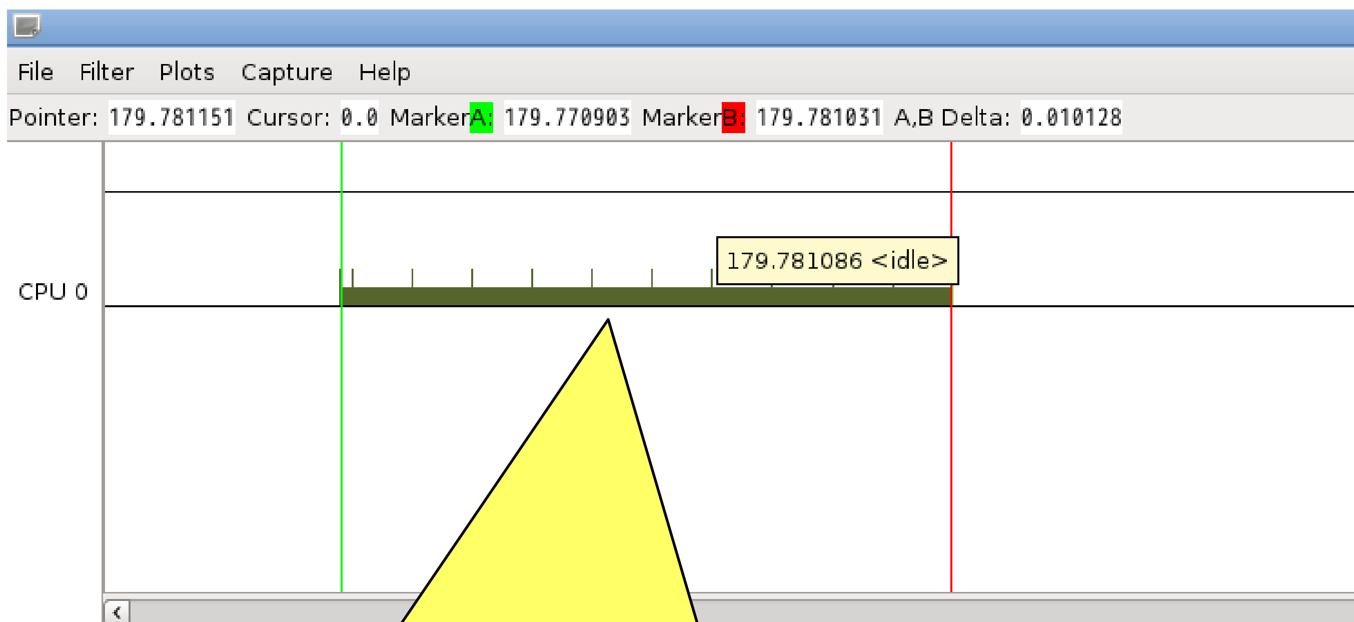


周期はusオーダの指定でもOK



実行時間の誤差は，同様に発生

周期30.0ms，実行時間10.5ms



実行時間の誤差は，同様に発生
ただ，10.5msの実行時間なら，
11回tickを必要とするので
実行時間が不足するというパターンは減る．

Outline

- Motivation
- SCHED_DEADLINEについて
- 評価
- **Conclusion**

Conclusion

- **SCHED_DEADLINEについて紹介**
- **評価・分析**
  - CPU利用率×周期の値として1ms以上あけておかないとNG
  - 実行時間と周期の時間の粒度が異なるせいで，厳密な意味でのデッドライン保障はできていない．
  - ms単位での周期や実行時間を持つタスクには誤差が大きい
- **今後**
  - 時間粒度をusオーダで指定できるように統一したい．
  - デッドライン保障ができているかどうかの検証ツールが必要
  - タスクの最悪実行時間見積りツールの導入
  - kernel v3.0への対応

TOSHIBA

Leading Innovation >>>

ご清聴ありがとうございました.

Copyright 2012, Toshiba Corporation.