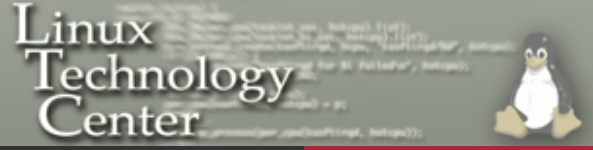


SystemTap How-to

2006/11/9

日立製作所システム開発研究所

Linux Technology Center

平松 雅巳 <*masami.hiramatsu.pt@hitachi.com*>

HITACHI
Inspire the Next

Contents

HITACHI
Inspire the Next

1 SystemTap Overview

HITACHI
Inspire the Next

Background Issues

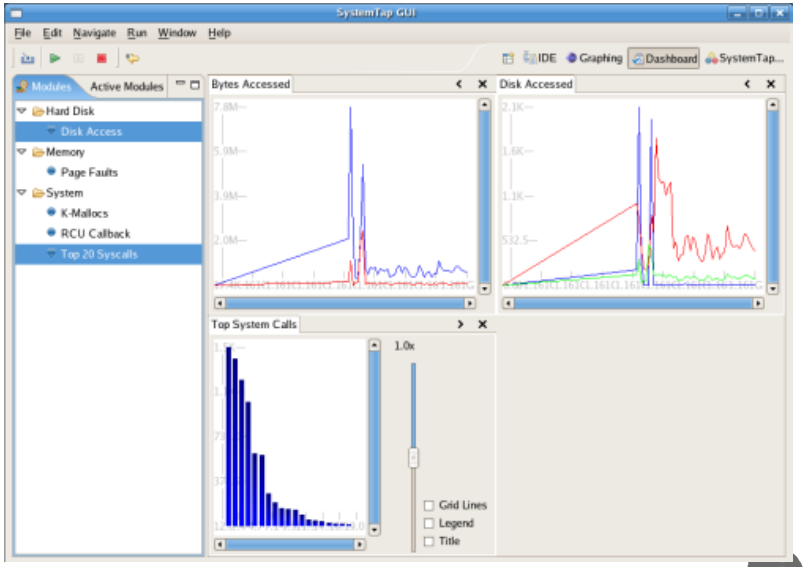
- 性能解析
  - システム全体の性能劣化を、一つのツールだけで解析したい
- デバッグ
  - カーネルのデバッグには、カーネルに対する詳しい知識が必要
- システム管理
  - Linuxは多種多様なので、カーネルコードに依存したツールは管理し難い
  - システムの状態を詳細に監視したい

Traditional Solutions

- ベンチマーク
  - ブラックボックス方式
  - 詳細な結果が出にくい
- Oprofile
  - 最良の方法の一つ
  - 「ある条件の下でだけおかしいコード」に対応しにくい
- Printk
  - 私も良く使います(^^;
  - カーネルの再構築が必要になる
- /proc, /sys
  - バージョンやコンフィグによって違いすぎる
  - 性能低下やバグの検出には使えない

SystemTap Introduction

- SystemTap
  - システム全体の状態を解析するためのツール
  - デバッグ情報を利用しているので、詳細な情報取得が可能
  - システムの状態を監視することも可能
  - スクリプトを使っているので改造が簡単。
  - マルチアーキテクチャ i386, x86-64, ia64, ppc64

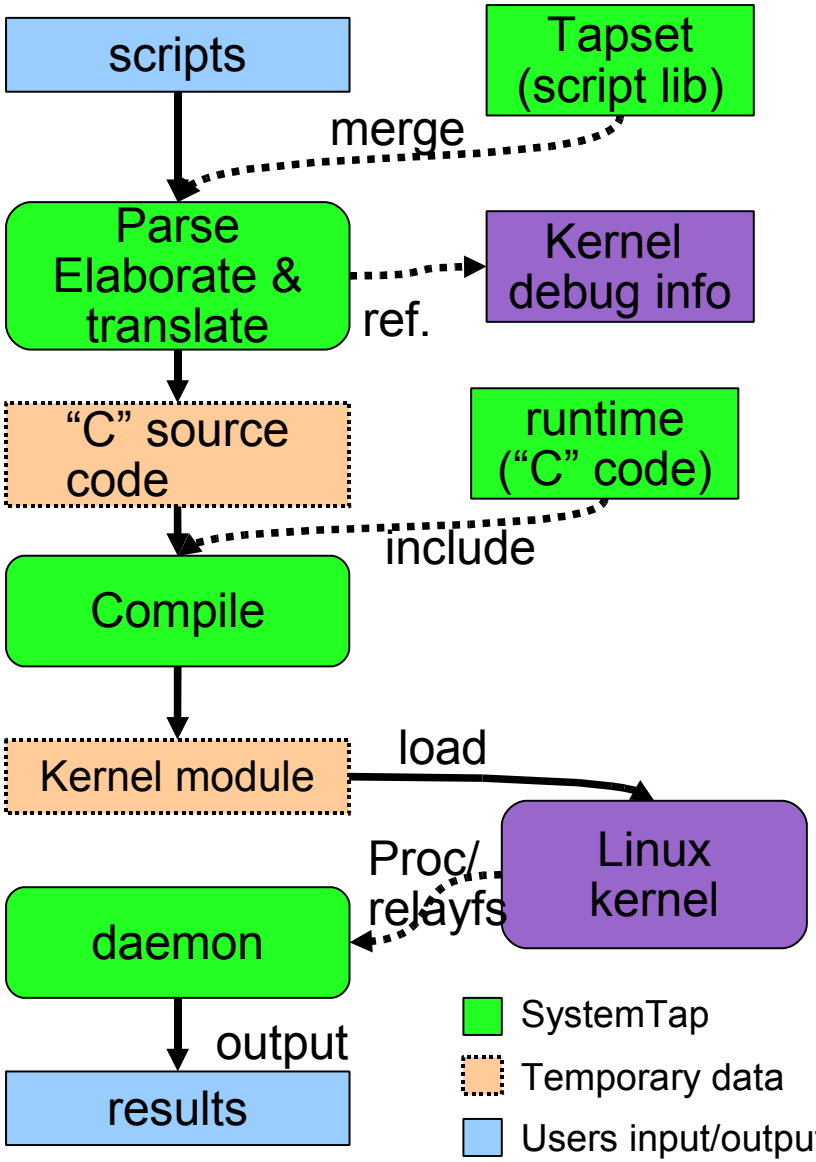


SystemTap Features

- 可用性
  - **パッチを当てずに**、動作中のカーネルから情報を取得できます
- 柔軟性
  - **スクリプト言語**で、簡単にお望みのプログラムを書けます
- 低負荷
  - インタプリタなど使わず、**ネイティブコード**を実行します
- 安全性
  - ページフォルト処理、無限ループのチェック、動的メモリ確保しないなど、**安全面を重視したデザイン**
  - 実際の運用システムで使えます

SystemTap Diagram

- 5つの処理段階
  - パース処理
  - 推敲処理
  - 変換処理
  - (最適化)
  - コンパイル(GCC)
  - 実行
- リソース
  - Tapset
    - スクリプトのライブラリ
  - Runtime
    - Cのライブラリ



SystemTap Project

- プロジェクトサイト
  - http://sourceware.org/systemtap/
- プロジェクトメンバー
  - Red Hat : コアプログラム
  - IBM : kprobes, スクリプト, トレーサ, GUI
  - Intel : kprobes, Scripts
  - 日立 : kprobe-booster, djprobe, トレーサ
- ディストリビューション
  - Red Hat Enterprise Linux, Fedora Core, Gentoo, SuSE Linux, Debian GNU/Linux, etc.
  - アップストリームカーネルでも使える

2
SystemTap on Your Linux

HITACHI
Inspire the Next

How to Install the SystemTap

- 必要なパッケージ
  - elfutils (latest version)
  - gcc > 4.0.2
  - kernel-devel, kernel-debuginfo
- Fedora Core / RedHatへのインストール方法
  - fedora-core.repoの編集
    - *core-debuginfoのセクションで、enable=1にしておく*
  - Yumでsystemtap, kernel-devel, kernel-debuginfoをインストール
    - *依存性は自動的に解消される（ただし、kernel-develやkernel-debuginfoは自動的に入らない）*
    - *FC6の systemtap は古いのでyumでupdateしておく*

How to Build the Kernel

- 最新カーネルで使う場合
  - SystemTapに必要なコンフィグ
    - Loadable module support
      - Enable loadable module support [Y]
      - Module unloading [Y]
    - General setup
      - Kernel->user space relay support [Y]
    - Instrumentation Support
      - Kprobes [Y]
    - Kernel hacking
      - Kernel debugging [Y]
      - Compile the kernel with debug info [Y]
  - インストール後、debug kernelへのリンクを張っておくこと

```
$ mkdir -p /usr/lib/debug/lib/modules/ker_ver/
$ ln -s /lib/modules/ker_ver/kernel
    /usr/lib/debug/lib/modules/ker_ver/
$ ln -s /lib/modules/ker_ver/build/vmlinux \
  /usr/lib/debug/lib/modules/ker_ver/
```

How to Use the SystemTap

- 基本的な使い方

  $ **stap** [options] <*script file*> | - | -e <*script*>

  - スクリプトをファイルや標準入力、コマンドラインから入力できる (“sed”のように)
  - sudoが使えるユーザなら使用可能

- 基本オプション
  - -v : メッセージレベル (スクリプトデバッグ用)
  - -b : relayを使う (大量のデータ転送用)
  - -I <DIR> : 追加のスクリプトをDIRから探す
    - 共通ルーチンをDIR以下で管理できる
  - -o <file> : 出力ファイルの指定
    - 通常は標準出力に出す

How to Use the SystemTap (cont 'd)

- EX)
  - "sample.log"ファイルへの出力

```
$ stap sample.stp –o sample.log
```

  - relayを使う

```
$ stap sample.stp –b
```

    - このとき、出力は"probe.out" ファイルに出る
  - コマンドラインから直接スクリプトを与える

```
$ stap –e "probe begin { printf(\"hello systemtap\")}"
```

  - スクリプトのデバッグ

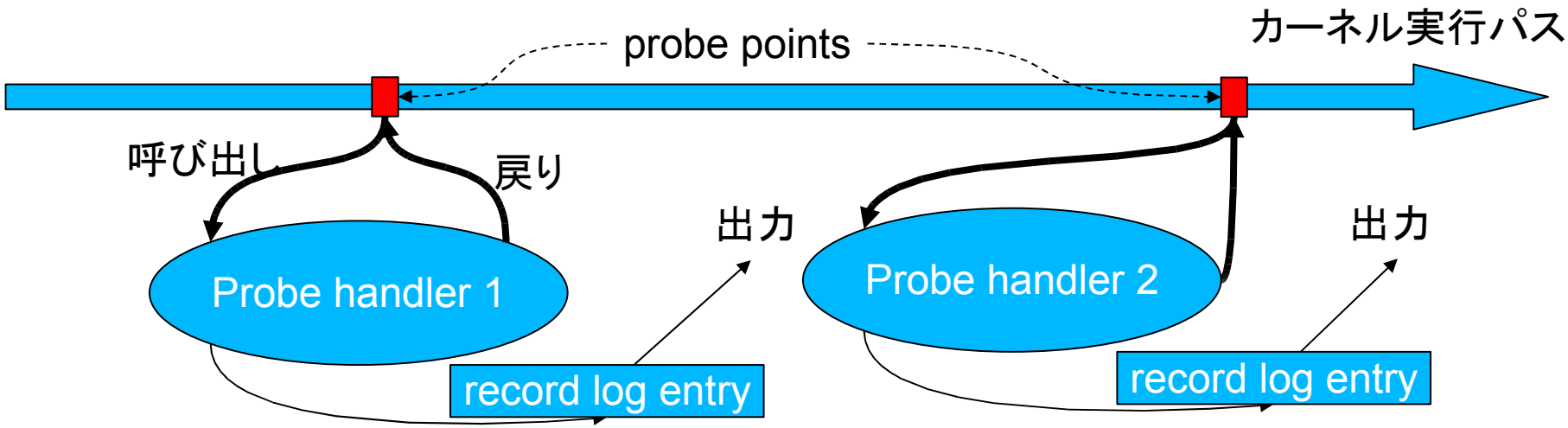
```
$ stap buggy.stp –vvv
```

3
Scripting How-to

HITACHI
Inspire the Next

Concept of Probing

- プローブとは
  - 実行中のカーネルのプローブポイントから情報を取得する



情報取得

ローカル変数
i, n
関数の引数
bio, nr_requests
グローバル変数
current

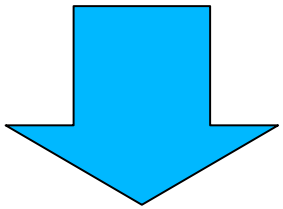
情報取得

ローカル変数
j
関数の引数
req
グローバル変数
current

Script Example1

- コンテキストスイッチを表示

```
probe kernel.function("__switch_to") {
        printf("switch %s to %s\n", execname(), kernel_string($next_p->comm))
}
```



```
$ stap switch_process_name.stp
switch gnome-terminal to stpd
switch stpd to X
switch X to xscreensaver
switch xscreensaver to X
switch X to gnome-terminal
switch gnome-terminal to X
```

Script Example1:Commentary

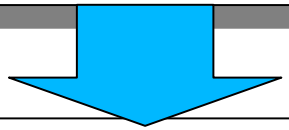
```
probe kernel.function("__switch_to") {
        printf("switch %s to %s\n", execname(), kernel_string($next_p->comm))
}
```

- **probe** *probepoint* **{** statements **}**
  - 指定されたprobepointでstatementを実行する
  - probepoints
    - kernel.function("関数名[@ファイル]")[.return]
    - kernel.module("モジュール名").function("関数名")
- **printf**(*"format"*, ...)
  - Cのprintfとほぼ同じ書式
- **$***struct***->***member*
  - カーネル変数へのアクセスには**$**をつける
  - 構造体のメンバへのアクセスには**->**を使う

Script Example2

- Hello World

```
global hello
probe begin {
        hello[1] = "hello"; hello[2] = "systemtap"
        hello[3] = "world"
        exit()
}
probe end {
        foreach([i] in hello)
                printf("%s\n", hello[i])
}
```



```
hello
systemtap
world
```

Script Example2:Commentary

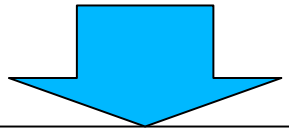
```
global hello
probe begin {
        hello[1] = "hello"; hello[2] = "systemtap"
        hello[3] = "world"; exit()
}
probe end {
        foreach([i] in hello)
                printf("%s\n", hello[i])
}
```

- global*グローバル変数*
  - スクリプト内のどこからでも参照可能
- 連想配列
  - 配列は全て連想配列
  - **foreach([インデックス変数] in** *連想配列***)**
- begin, end
  - スクリプトの開始と終了(exit())で呼ばれる

Script Example3

- プロセスのスケジュール頻度プロファイル

```
global top
probe timer.ms(1) {
        top[execname(), pid()]++;
}
probe timer.ms(1000) {
        foreach([i, p] in top-)
                printf("%s(%d) = %d\n", i, p, top[i,p])
        delete top
}
```



```
swapper(0) = 440
troff(4462) = 17
Xorg(2108) = 17
grotty(4463) = 9
nroff(4453) = 4
```

Script Example3:Commentary

```
global top
probe timer.ms(1) {
        top[execname(), pid()]++;
}
probe timer.ms(1000) {
        foreach([i, p] in top-)
                printf("%s(%d) = %d\n", i, p, top[i,p])
        delete top
}
```

- Timerプローブポイント
  - 一定周期ごとに実行される
- 連想配列の拡張
  - 多次元配列も扱える
  - foreach(インデックス**[+|-]** in 連想配列**[+|-]**)
    - 値やインデックスでソート
  - **delete** *配列*
    - 配列の中身をクリア

Demonstrations

- Function Call Tracer
  - サブシステムの関数の呼び出しをトレース
- Profiling
  - サンプリングベースのプロファイリング
- Disk I/O operation
  - gnuplotで表示

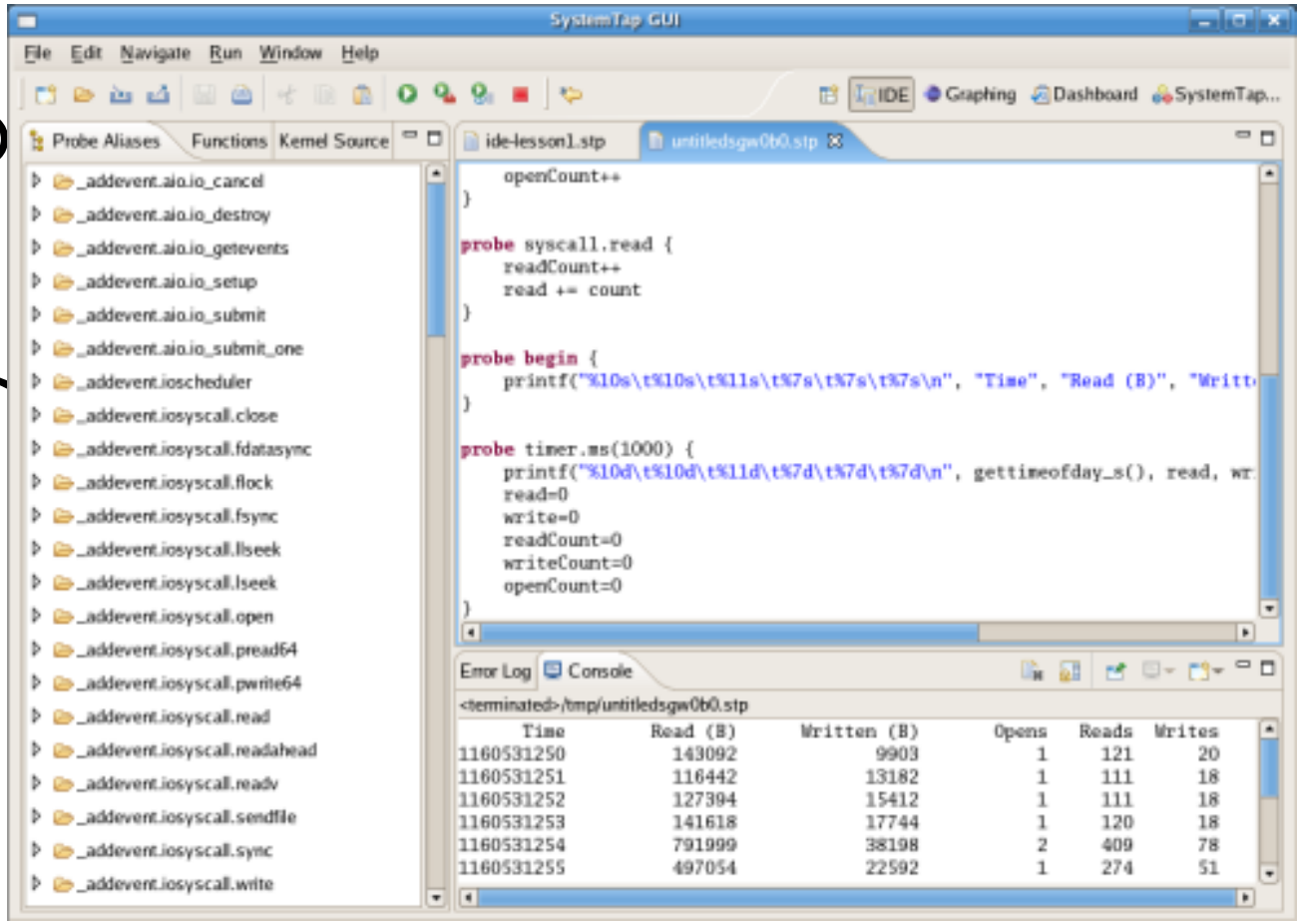
- →参照元
  - http://sourceware.org/systemtap/wiki

SystemTap GUI

- EclipseベースのIDE
  - スクリプトの開発を補助
- セットアップ
  - 最新のビルド済みパッケージのダウンロード
    - http://stapgui.sourceforge.net
  - パッケージの展開
  - 起動（インストール不要）
    - ただし、初回起動時にIDE Perspectiveを表示すると、全てのtapsetsをパースしようとする（数十分かかる）
    - *反応がなくなるが、ハングアップしているわけではない。*
- FC6の注意点
  - /etc/sudoersの変更 (一般ユーザから使いたいとき)
    - requirettyを”off”にしておく

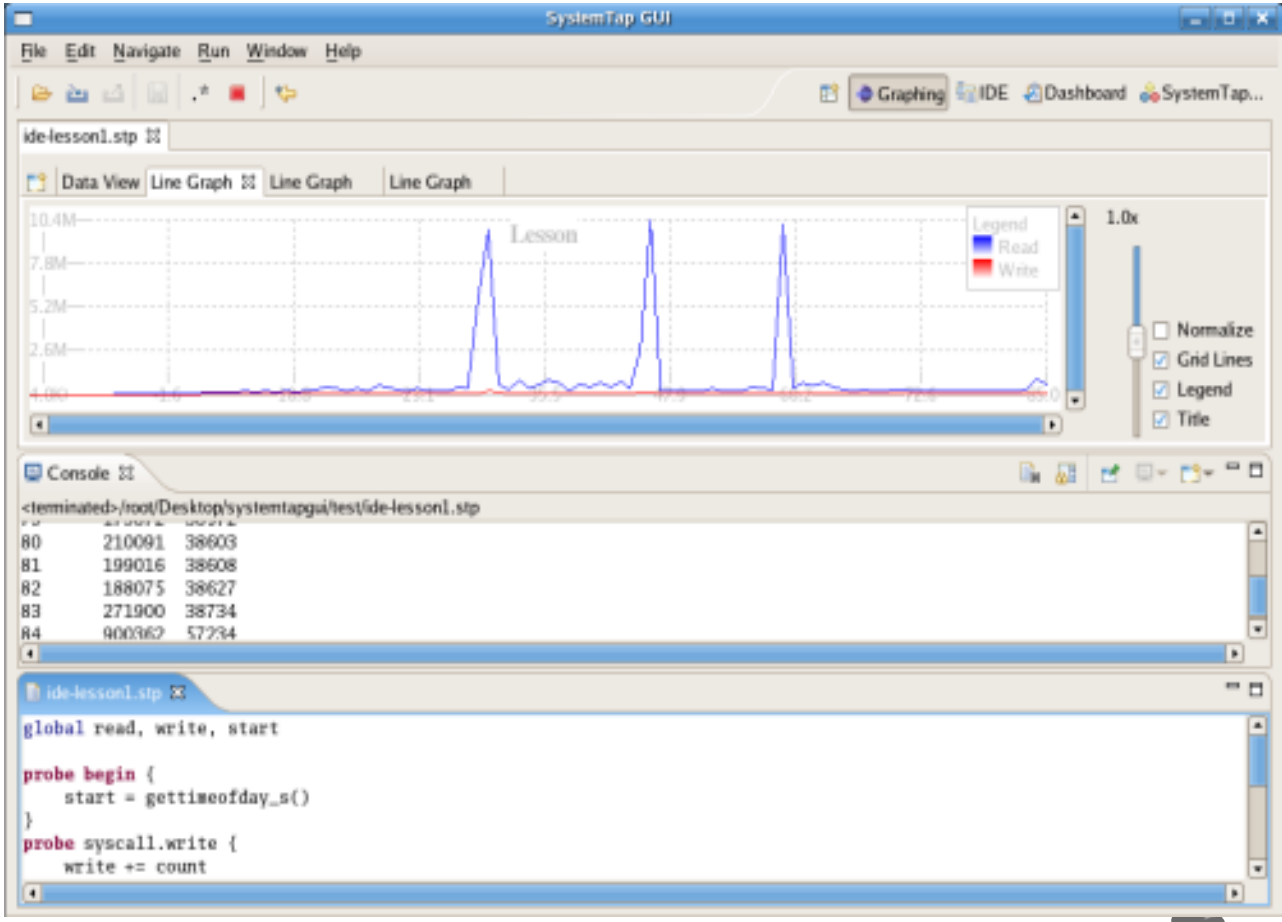
IDE Perspective

- EclipseベースのIDE画面
  - 自動補完
  - エイリアスのカタログ
  - 利用可能な変数のヒント



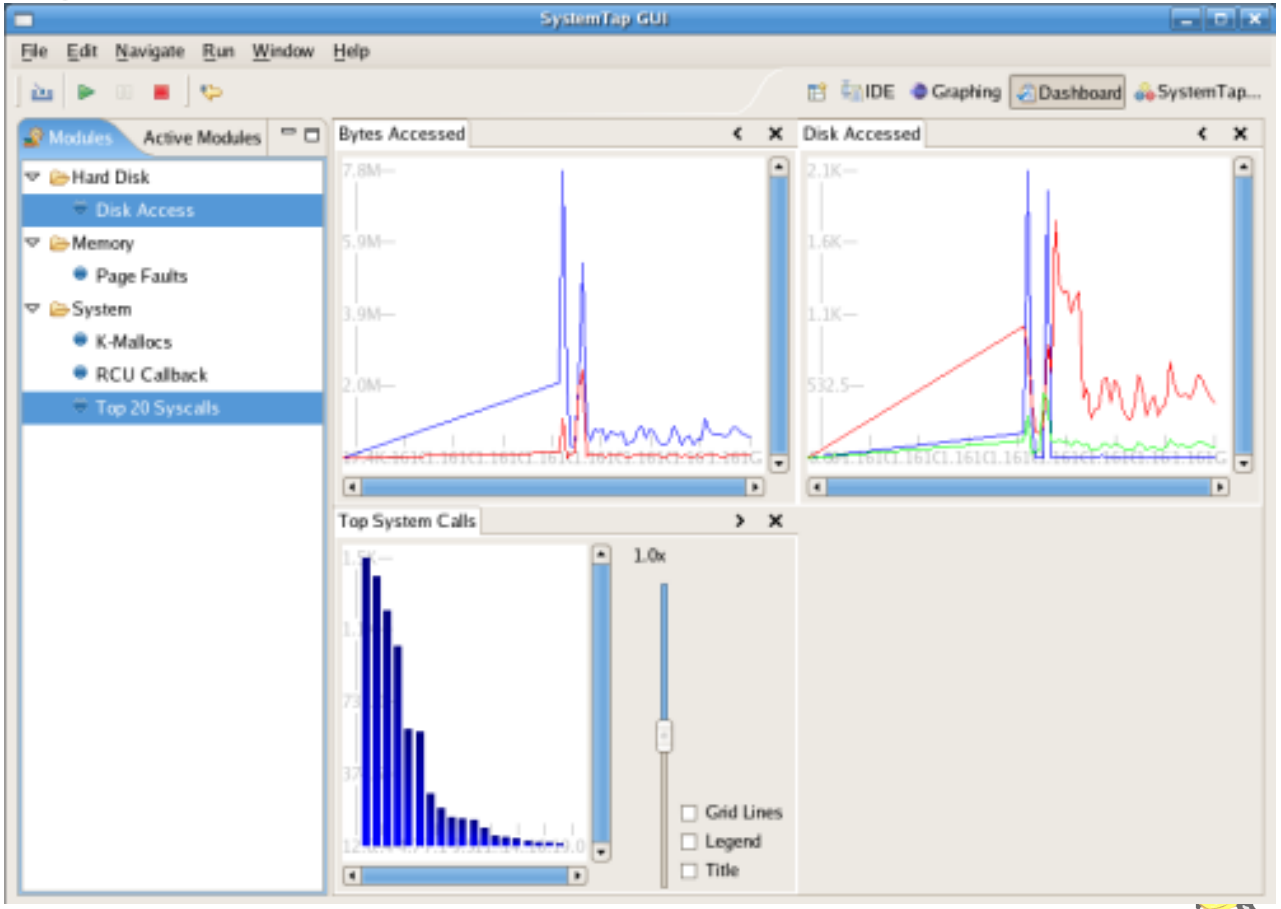
Graph Perspective

- グラフ画面
  - 実行中に更新
  - 複数のグラフを描画
  - フィルタ可能



Dashboard Perspective

- スクリプトパッケージのコレクション
  - 直接実行する
  - 詳細なシステム情報
  - 簡単に実行可能



Demonstration

- SystemTapGUIのデモ

4
Performance Enhancement

HITACHI
Inspire the Next

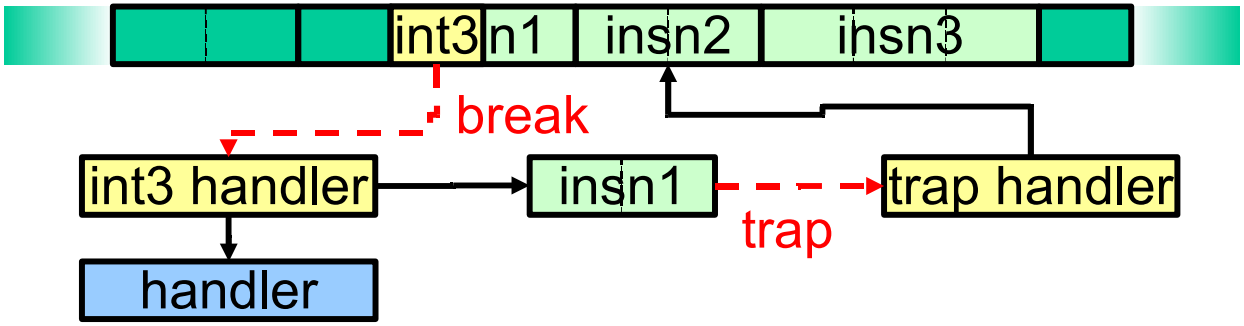
Overhead Issues

- SystemTapの使用例
- システム管理者: カーネルの性能解析
  - **課題: SystemTapによる性能劣化**
  - **劣化した性能**を測定しても意味がない。
- 企業ユーザ: カーネルのフライトレコーダ
  - **課題:SystemTapによる性能劣化**
  - トレーサは**常に有効**にしないといけない。
    - パニックの後起動したのでは遅すぎる
- 目標性能:
  - 50,000 probes/秒 の環境で 1%のオーバヘッド(200ns)

→ **Kprobe-booster** と **Djprobe** を開発

Kprobe

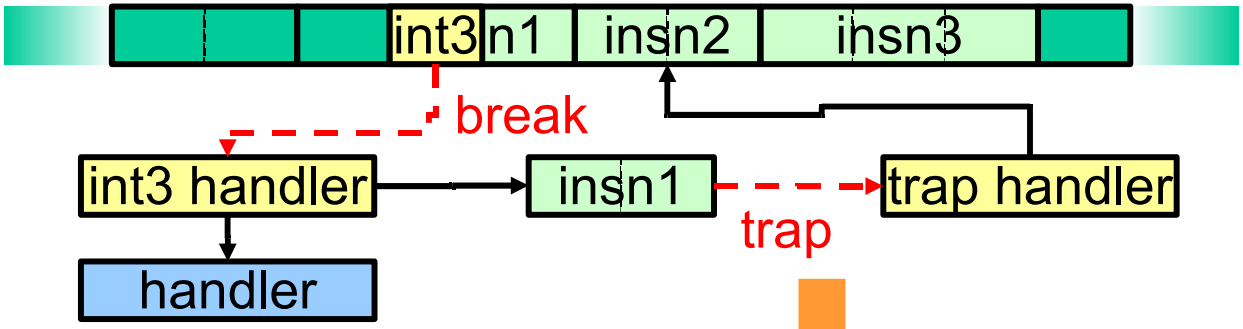
- コピーした命令をシングルステップ実行
  - kprobeは2回例外を使う(breakとtrap)



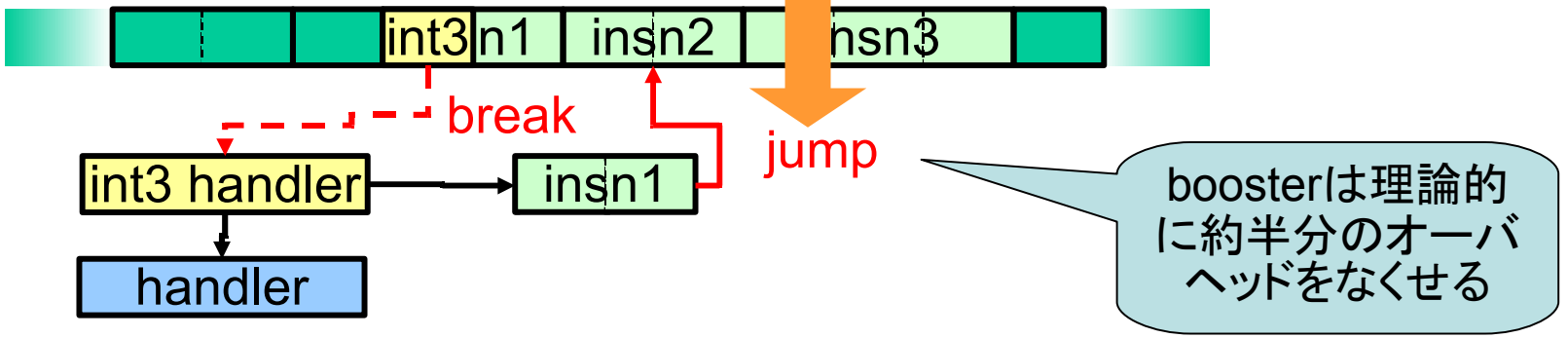
  - 例外命令の処理は実行に時間がかかる（～500ns）
  - 出来るだけ例外命令を減らしたい

Kprobe-booster

- コピーした命令を直接実行
  - kprobeは2回例外を使う(breakとtrap)

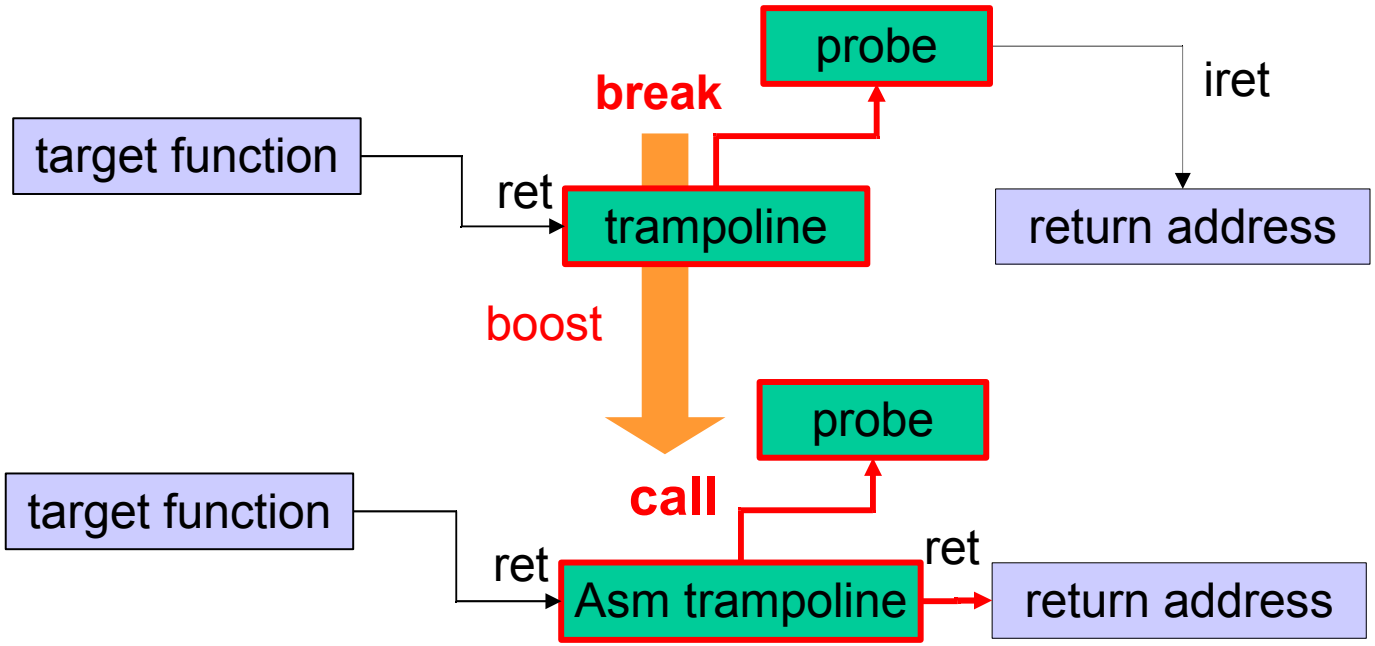


  - kprobe-booster はtrap例外を減らす



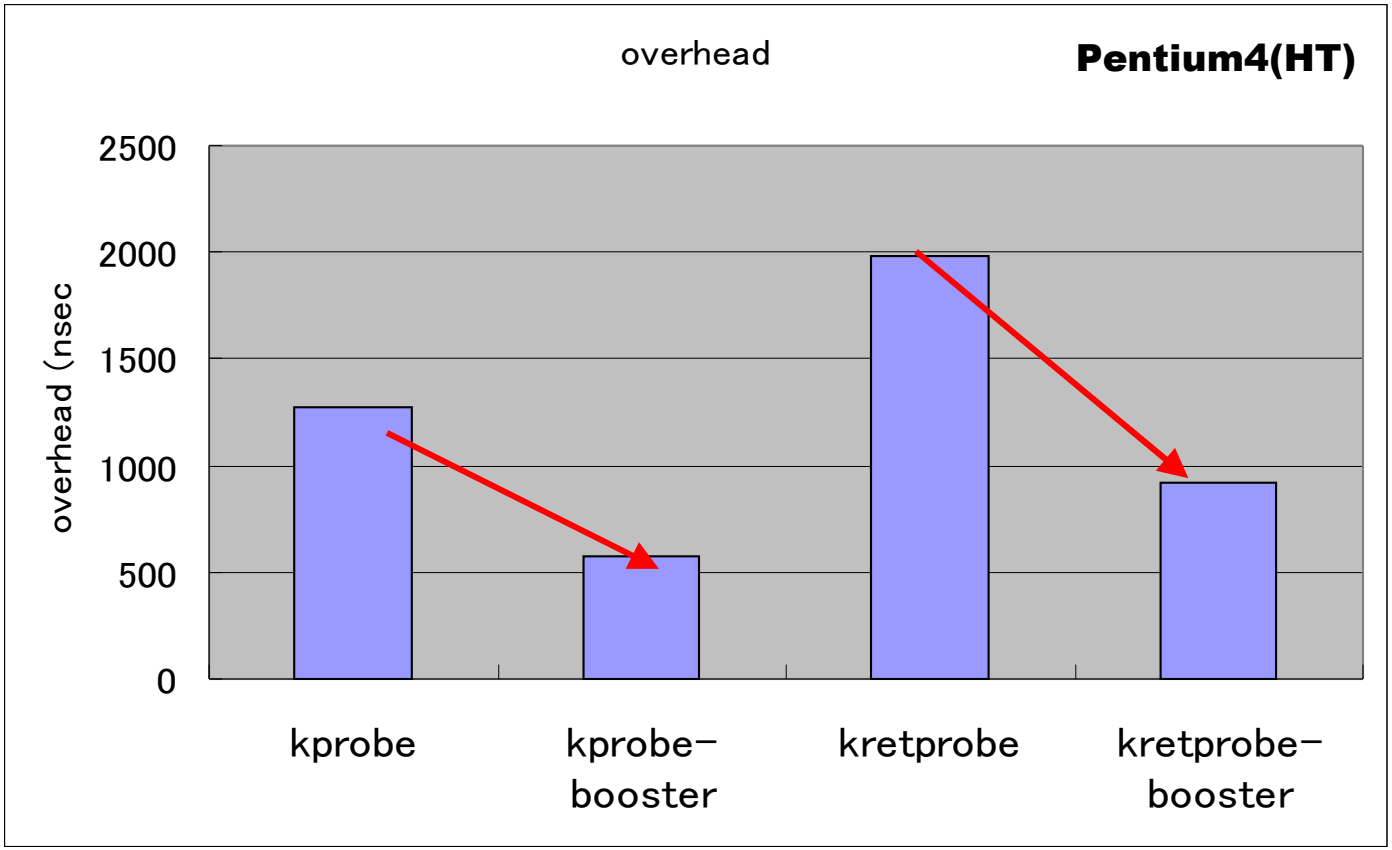
Kretprobe-booster

- Kretprobe
  - 関数の**戻りアドレスを書き換え**てkprobeを実行
- Kretprobe-booster
  - トランポリンコードの最適化
  - kprobeのbreak例外をcall命令に置き換え



Overhead: boosters

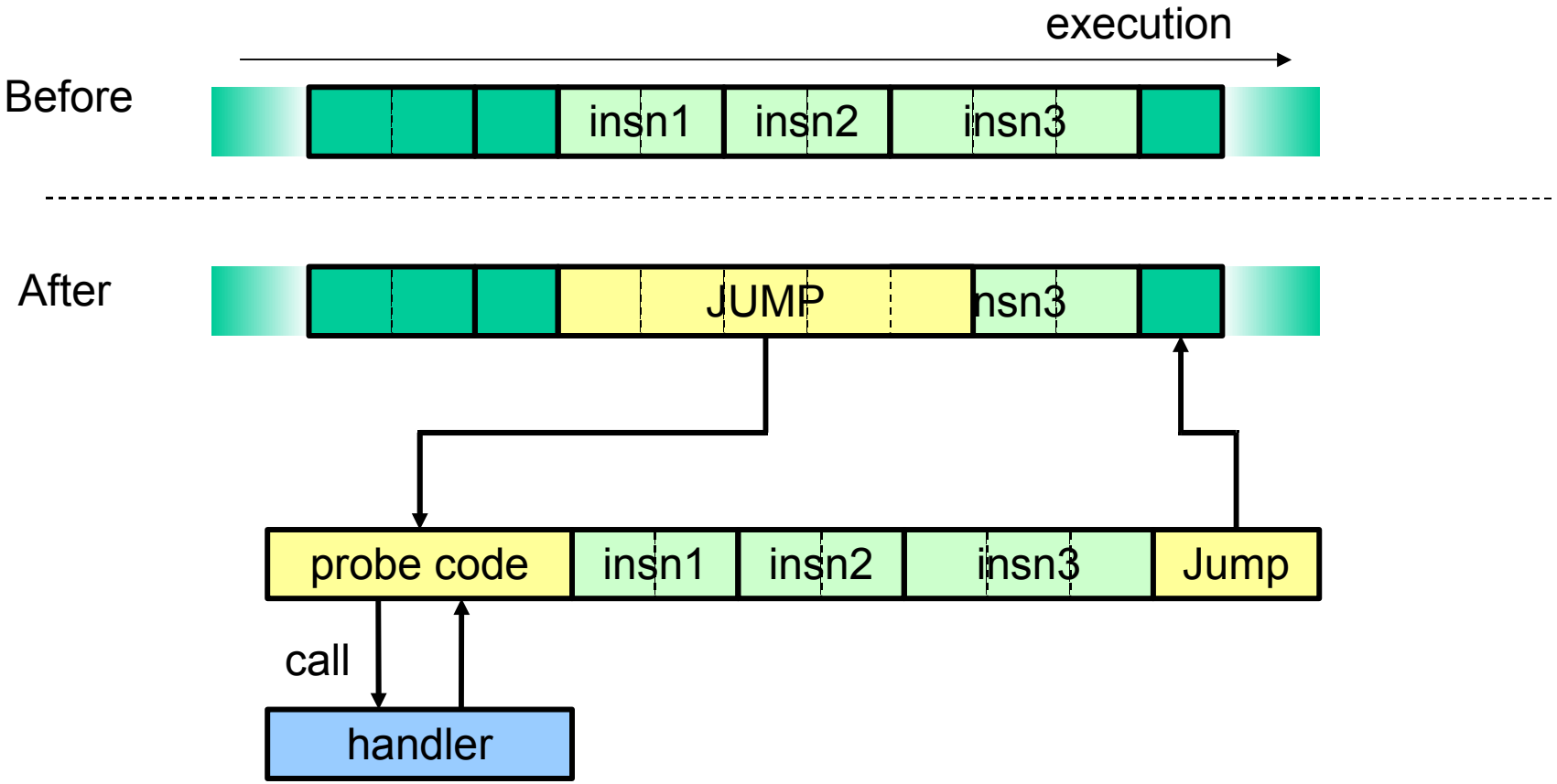
- プローブのオーバヘッド (プローブのみ)



オーバヘッドは半分に

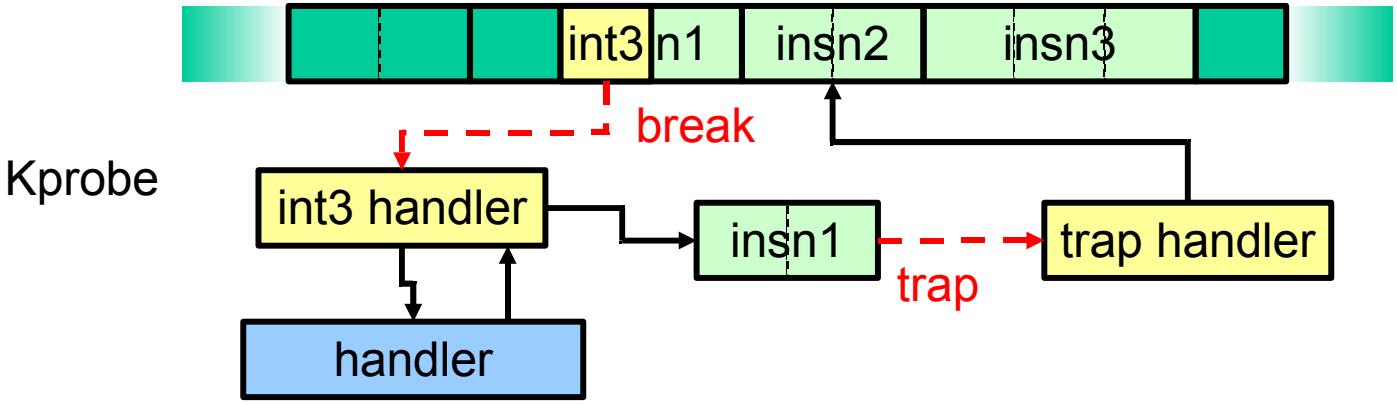
Djprobe

- Djprobeは、プローブコードに直接ジャンプする

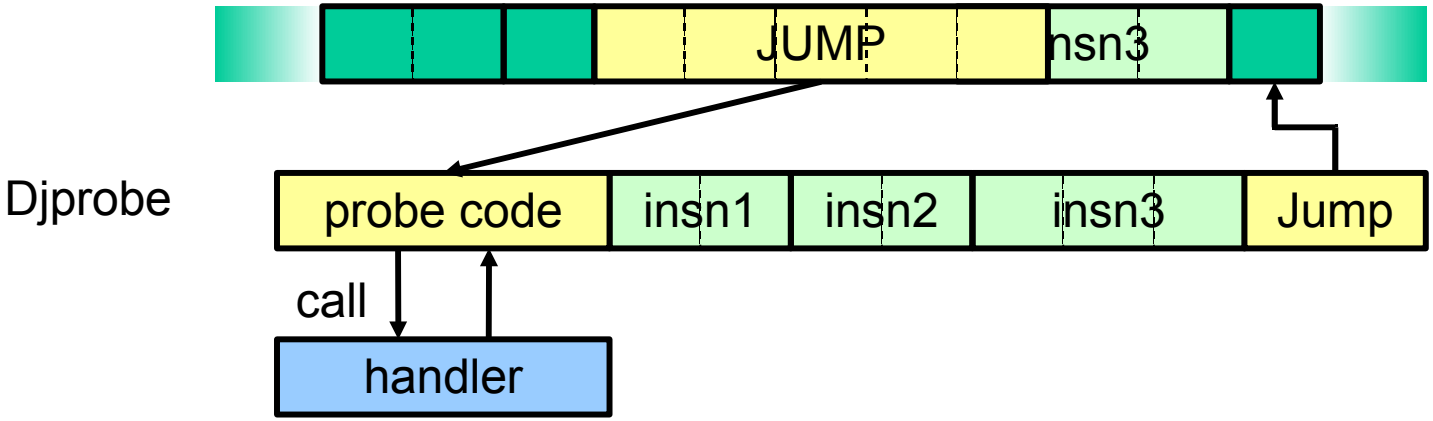


Comparing Djprobe with Kprobe

- Kprobeは2つの例外を実行

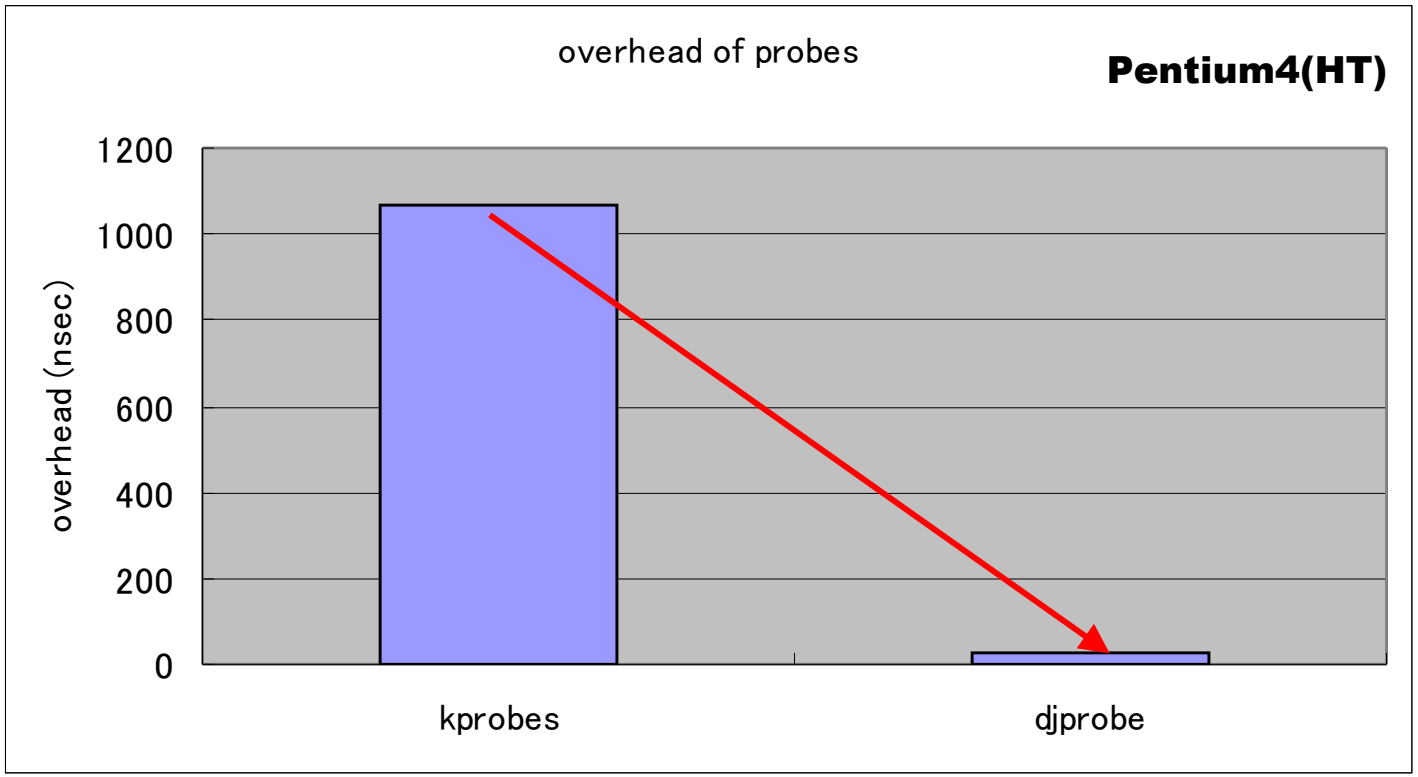


- Djprobeは例外を実行しない



Overhead: Djprobe

- プローブのオーバヘッド (プローブのみ)



Djprobeのオーバヘッドは**非常に小さい**(50~100ns)

5 Conclusions

HITACHI
Inspire the Next

Conclusion

- SystemTap
  - システム全体の問題を解析するのに有効なツール
  - カーネルから直に情報を得られる
  - プログラム可能な柔軟性
  - オンデマンドで利用可能
  - GUI（IDE）も存在
- Kprobe-booster/Djprobe
  - 極めてよいパフォーマンス

Future work

- これからの開発項目
  - ユーザ空間のプローブへの対応
  - ドキュメントの充実
  - オフラインモジュール対応
  - 解析ツールの開発
  - Djprobeのサポート
    - Kprobeのインタフェースと統合中
    - サポートアーキテクチャを増加
  - RAS強化
    - 企業サーバ向けRASトレーサ機能の開発

References

- 主な参照先
  - SystemTap Project
    - http://sourceware.org/systemtap/
  - SystemTap Wiki
    - http://sourceware.org/systemtap/wiki
  - SystemTap GUI
    - http://stapgui.sourceforge.net/
  - Djprobe
    - http://lkst.sourceforge.net/djprobe.html

Thank you!

and

Happy Stapping!

Trademarks

- LinuxはLinus Torvaldsの登録商標です。
- Pentiumは米国Intel co.の登録商標です。
- その他の記載されている社名および製品名は各社の登録商標また商標です。

Q & A