

open



USE



IMPROVE



EVANGELIZE

OpenSolaris HotTopics #3 Solaris System Performance Analysis Methodology

Hisayoshi Kato
Sun Solution Center

<http://blogs.sun.com/katohisa>

Sun Microsystems, K.K.
Tokyo, Japan

開
放
的
열린
مفتوح
libre
मुक्त
ಮುಕ್ತ
livre
libero
ముక్త
开放的
açık
open
nyílt
•••••
πικρ
オープン
livre
ανοικτό
offen
otevřený
öppen
открытый
வெளிப்படை

セミナーの内容

セミナーでカバーすること

- ✓ Solaris システムパフォーマンス解析方法の理解
- ✓ システムパフォーマンス解析に必要なツールの紹介

セミナーでカバーしないこと

- ✓ アプリケーションチューニング
- ✓ 基本的な DTrace の利用方法

Solaris システムパフォーマンスとは？

- ✓ Solaris からみると、ミドルウェアを含め Solaris アプリケーション ...
- ✓ ハードウェアからみれば、Solaris は、1つのソフトウェア ...
- ✓ システムパフォーマンスとは、アプリケーションが主導 ...

アプリケーション

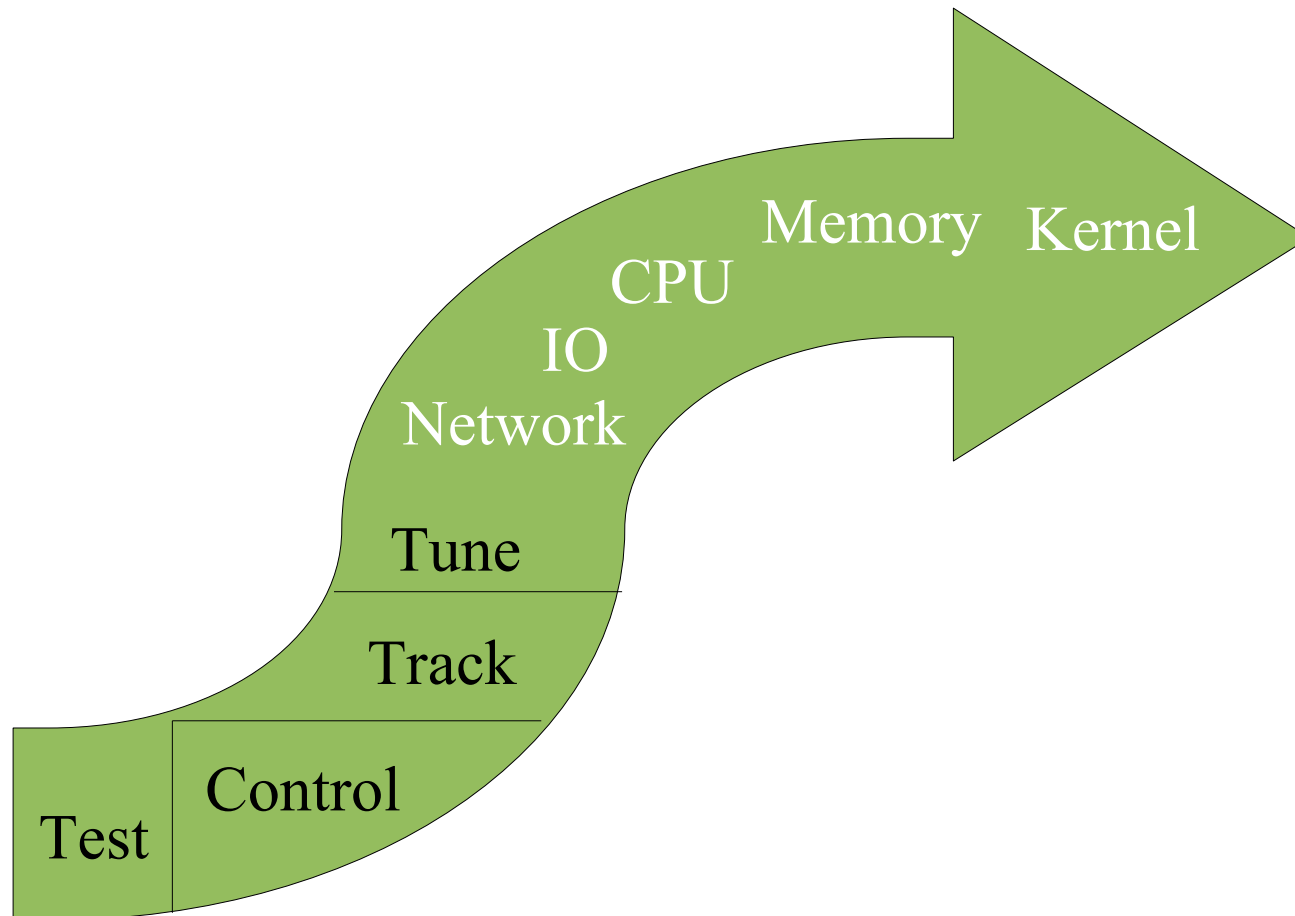
データベース /
アプリケーション
サーバー

オペレーティング
システム (Solaris)

ハードウェア

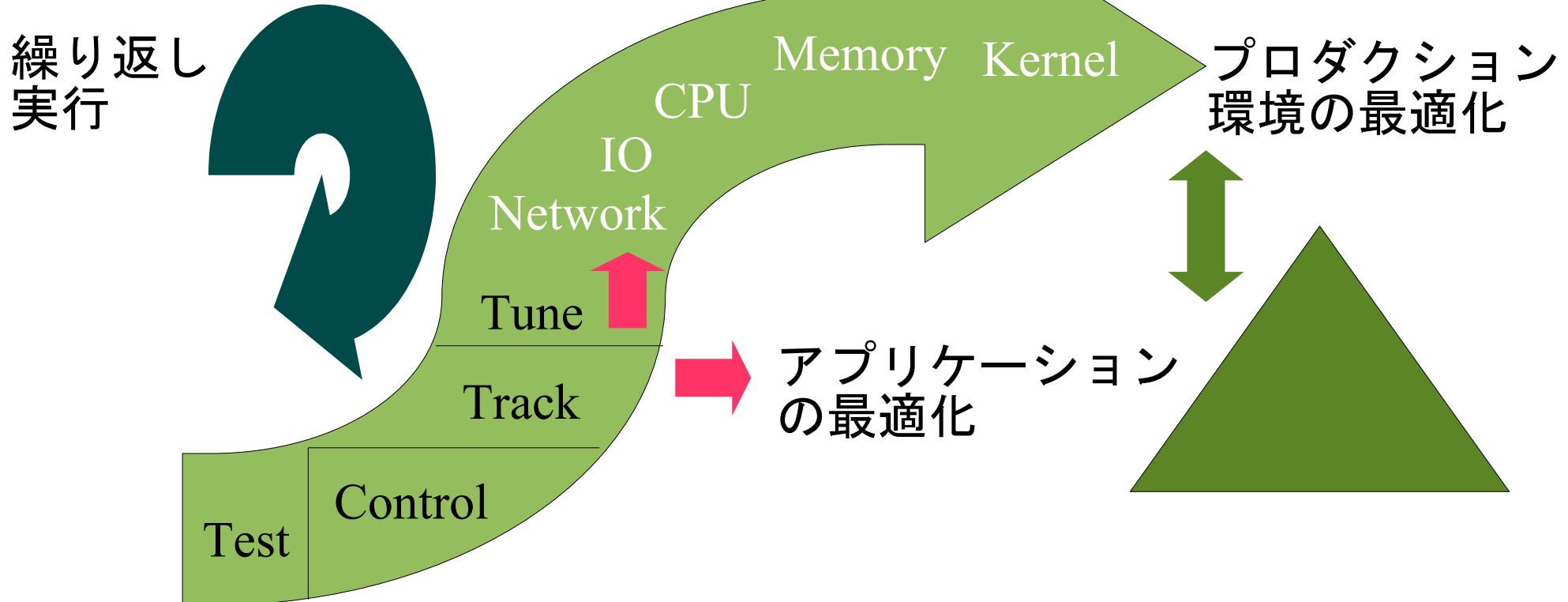
CPU
メモリ
ディスク
ネットワーク

アプリケーション分析の流れ

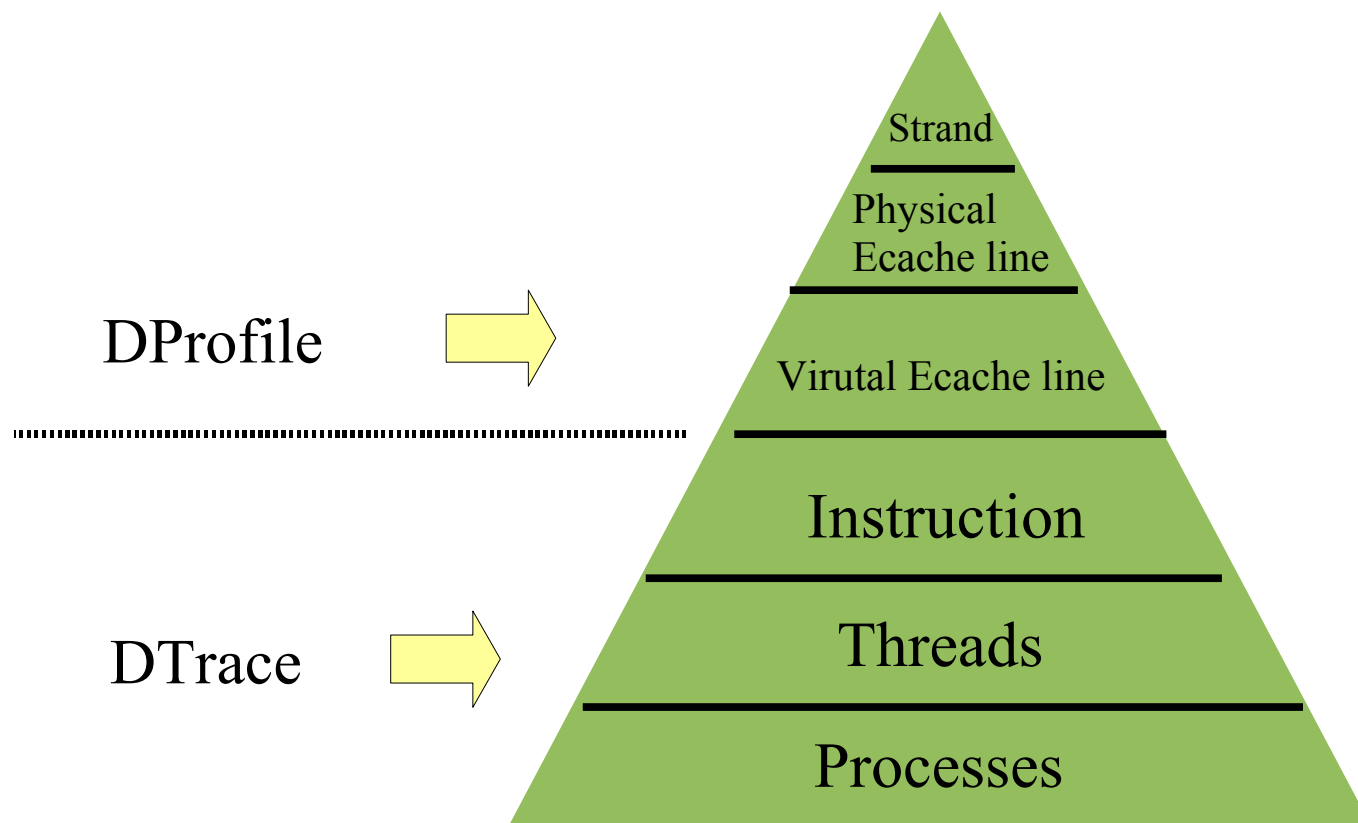


パフォーマンス解析ステップ

1. Test 構成チェック、HW 帯域テスト (ベンチマークテストなど)
2. Control アプリケーションの動作のチェック
3. Track エラーがないか確認、アプリケーションの最適化
4. Tune チューニング



アプリケーションの最適化



パフォーマンスツール

✓ Process stats

- ✓ cputrack プロセスごとの HW counters
- ✓ pargs プロセスの引数の確認
- ✓ pflags プロセスフラグの確認
- ✓ pcred プロセスクレデンシャルの確認
- ✓ psig プロセスシグナルの配置
- ✓ pstack プロセススタックの確認 ✓
- ✓ pmap プロセスマップの確認
- ✓ ptree プロセスツリーの確認
- ✓ ptime プロセスタイムの確認
- ✓ pwdx プロセスディレクトリの確認
- ✓ plockstat プロセス競合の確認

Process Tracing

- ✓ abitrace ABI の追跡
- ✓ dtrace プロセスの解析
- ✓ mdb ライブプロセスの解析
- ✓ truss システムコールとユーザ関数の解析

✓ System Stats

- ✓ acctcom プロセスアカウントティング
- ✓ busstat バス HW counters
- ✓ cpustat CPU hardware counter
- ✓ iostat IO&NFS) 統計
- ✓ kstat カーネル統計の分析
- ✓ mpstat プロセッサの統計
- ✓ netstat ネットワークの統計
- ✓ nfsstat nfs の統計
- ✓ trapstat trap の解析
- ✓ intrstat 割り込みの解析
- ✓ sar システム統計
- ✓ vmstat 仮想メモリの統計

✓ Process control

- ✓ pgrep プロセス grep
- ✓ pkill プロセスリストを kill ✓
- ✓ pstop プロセスを止める
- ✓ prun プロセスを開始
- ✓ prctl プロセスリソースをみる
- ✓ pwait プロセスをウェイトさせる
- ✓ preap ゾンビプロセスを取り除く

Kernel Tracing

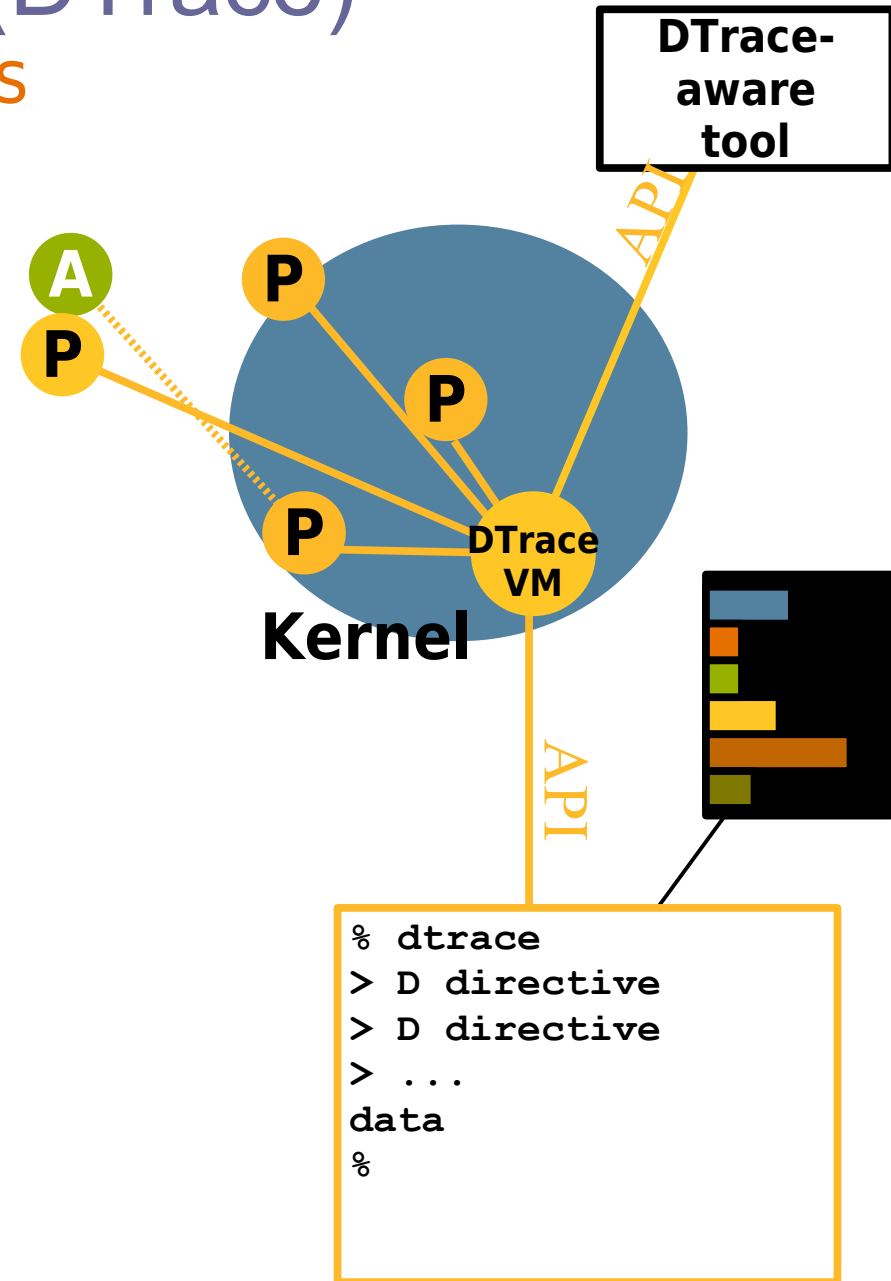
- ✓ dtrace カーネルの追跡と監視
- ✓ lockstat ロック統計の監視
- ✓ lockstat -k カーネルのプロファイル
- ✓ mdb ライブプロセスとカーネルのデバッグ

✓ Test Tools

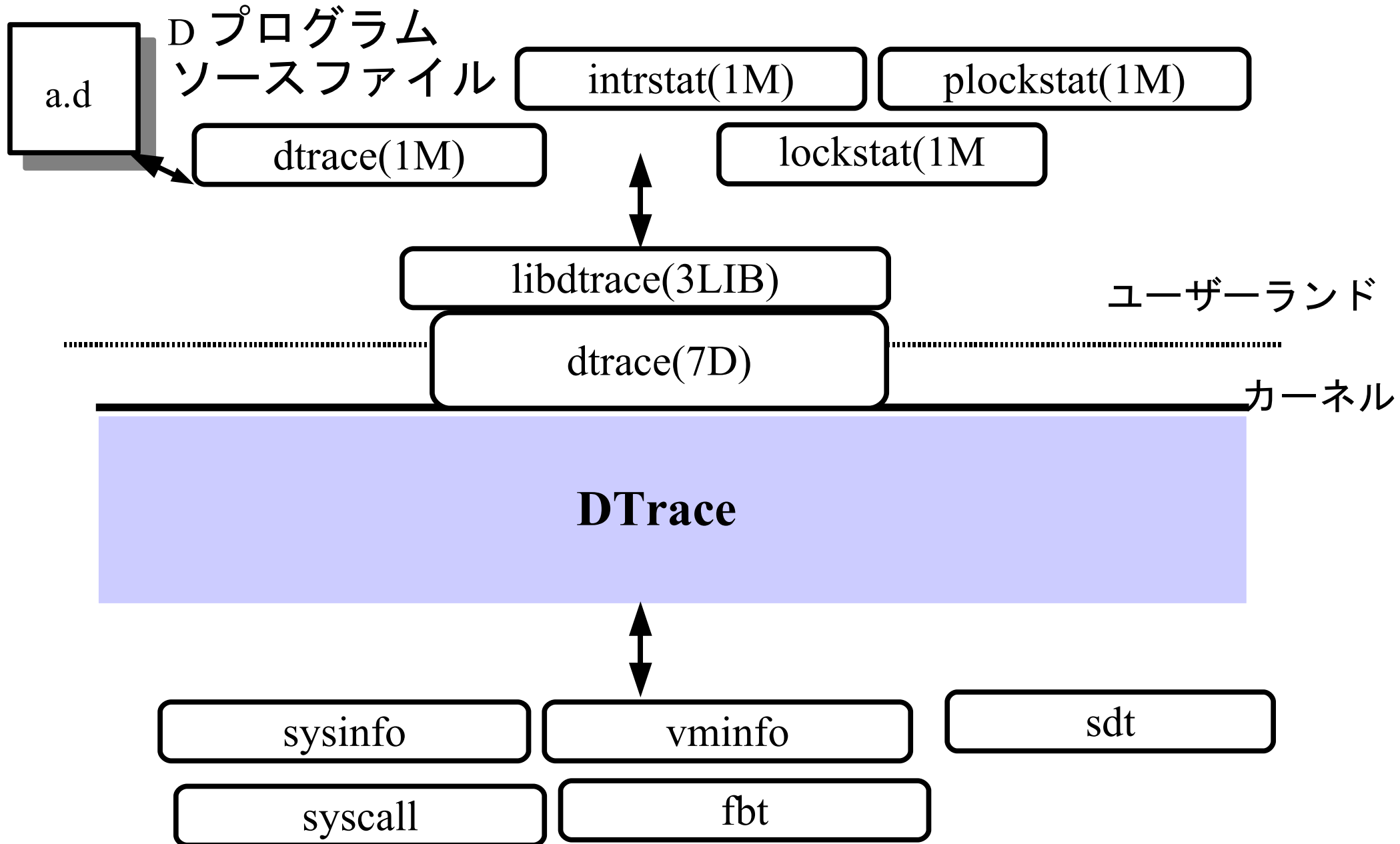
Solaris Dynamic Tracing (DTrace)

Designed for Production Systems

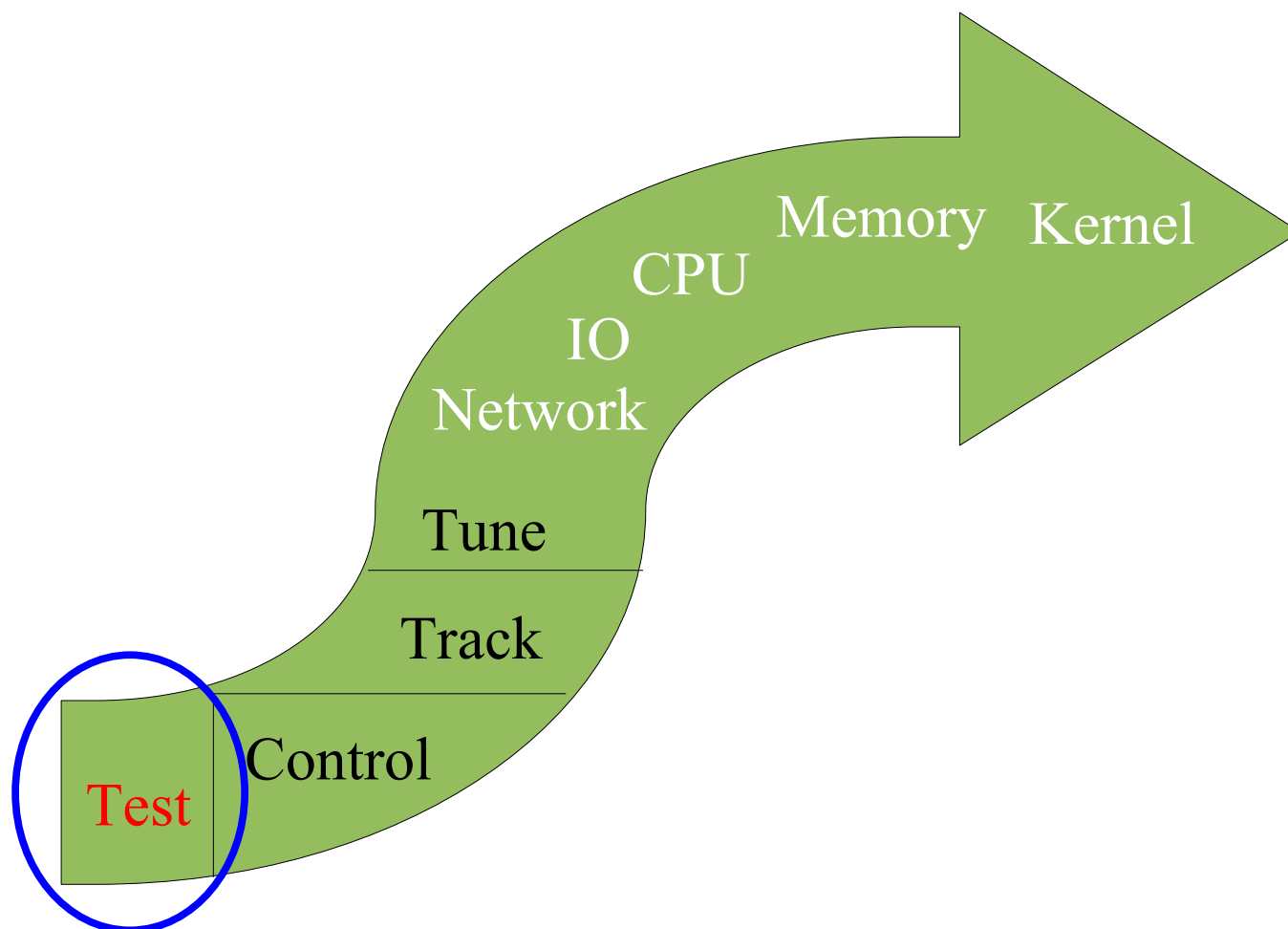
- ✓ パフォーマンスほぼ劣化なし
- ✓ アプリの改変なし
- ✓ アプリケーションから kernel まで
- ✓ システム全体を観測
 - ✓ 包括的に観測
 - ✓ スクリプト化
- ✓ パフォーマンスを詳細に解析
- ✓ 稼働中のシステムで実行可能



DTrace アーキテクチャ



アプリケーション分析の流れ

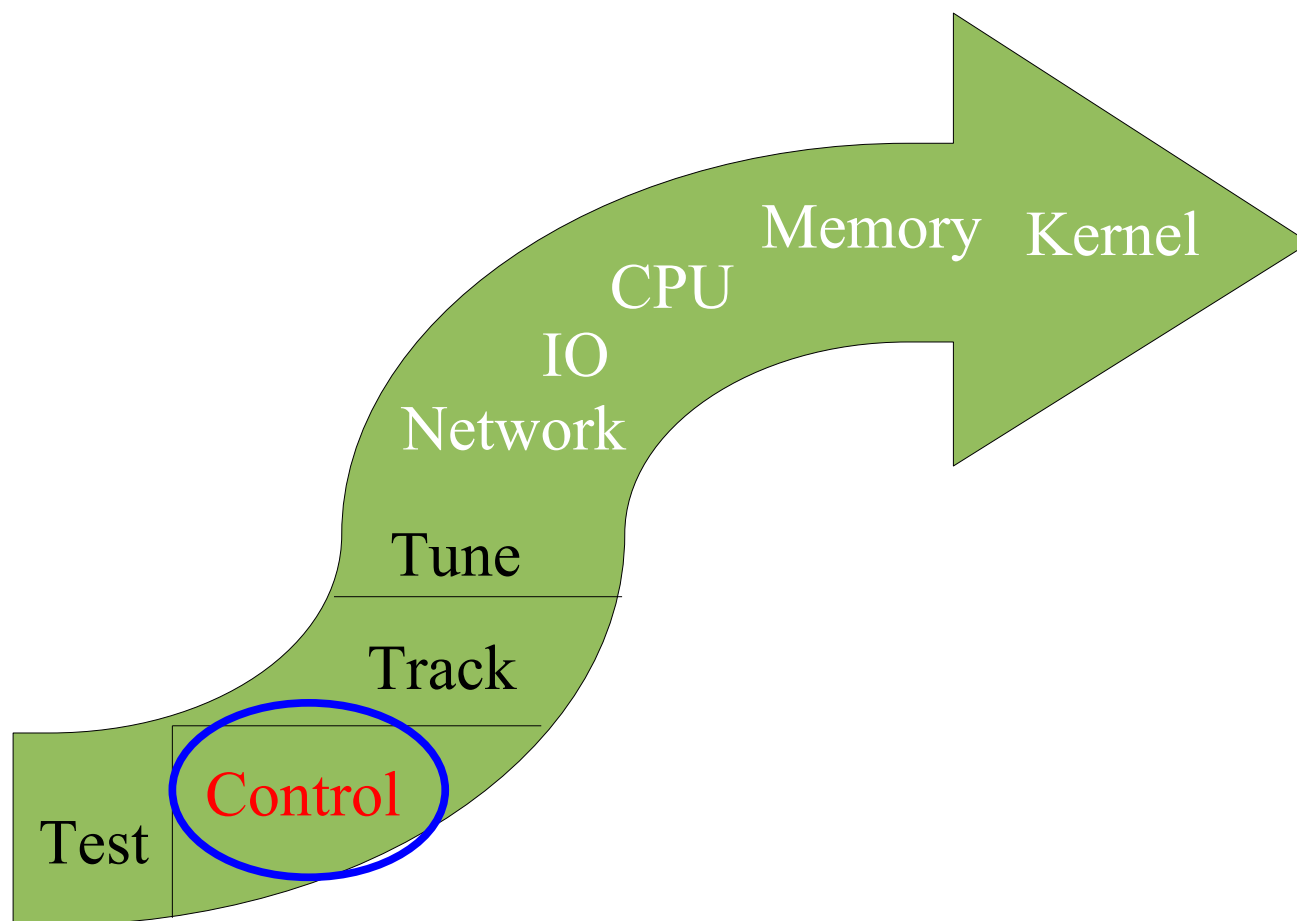




Test

- ✓ ハードウェアの準備
- ✓ 構成のチェックからはじめる。
 - ✓ prtdiag(1M) など
 - ✓ OpenSolaris bug database
 - ✓ サポート契約をされていれば、sunsolve へ
- ✓ 理論的なスループット／キャパシティを決定
- ✓ Test
 - ✓ Network: uperf など
 - ✓ IO: filebench など
 - ✓ CPU & RAM: libmicro など
 - ✓ SMP で設計されたシステムは、疑似テストで、スケラビリティをだせるか？チェックする。基準値、比較値があるとよい。

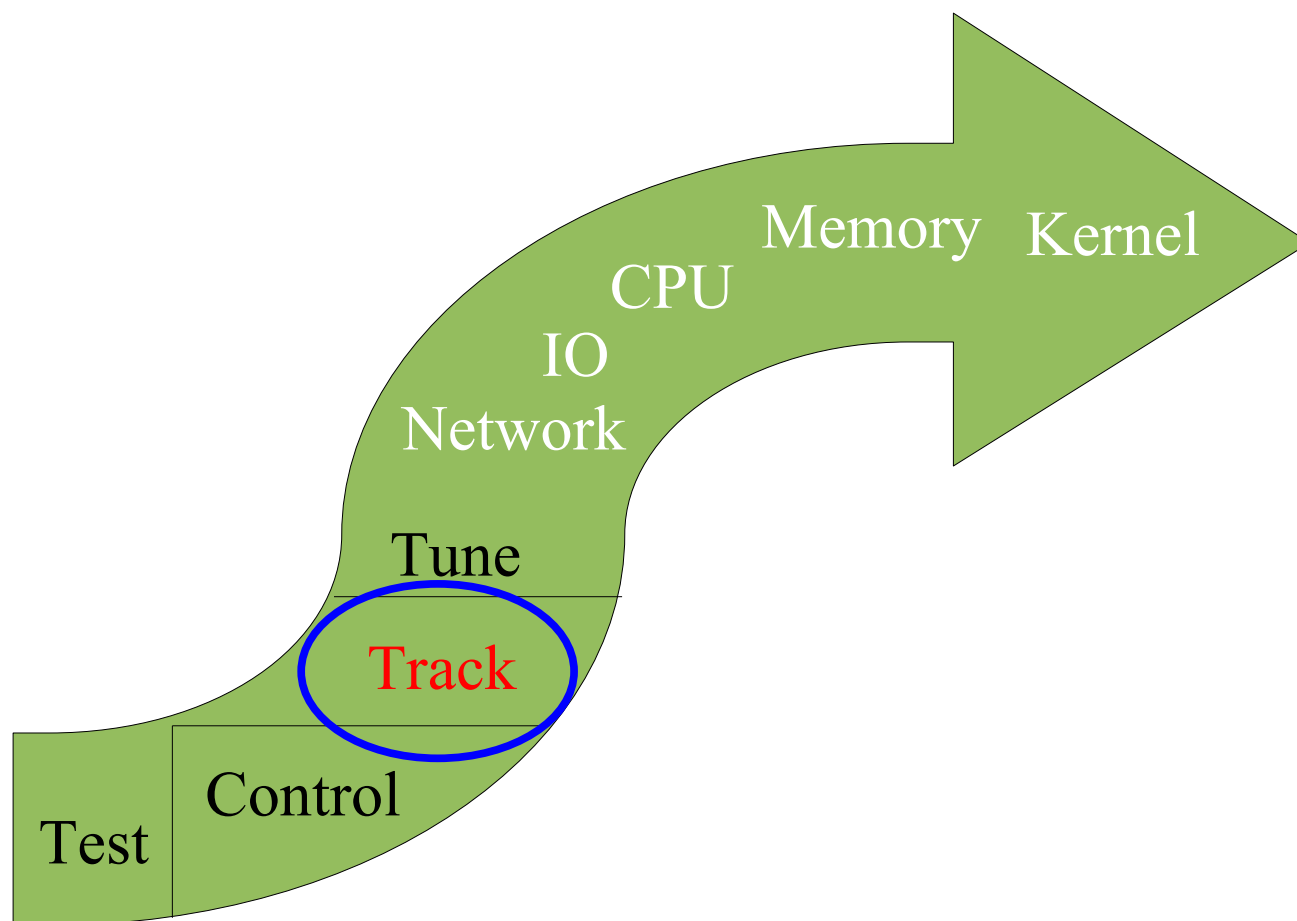
アプリケーション分析の流れ



Control

- ✓ アプリケーションの特性の把握
- ✓ プロセスの動作を確認
 - ✓ ps & prstat だけでなく、dtrace でも確認
 - ✓ `dtrace -n 'proc:::exec{printf("%s execing %s,uid/zone = %d/%s\n",execname,args[0],uid,zonename)}'`
- ✓ コンパイラの確認
 - ✓ シンボルテーブルがストリップされていないなら、下記で検出
 - ✓ `dump -c $q | grep "WS6U2"`

アプリケーション分析の流れ

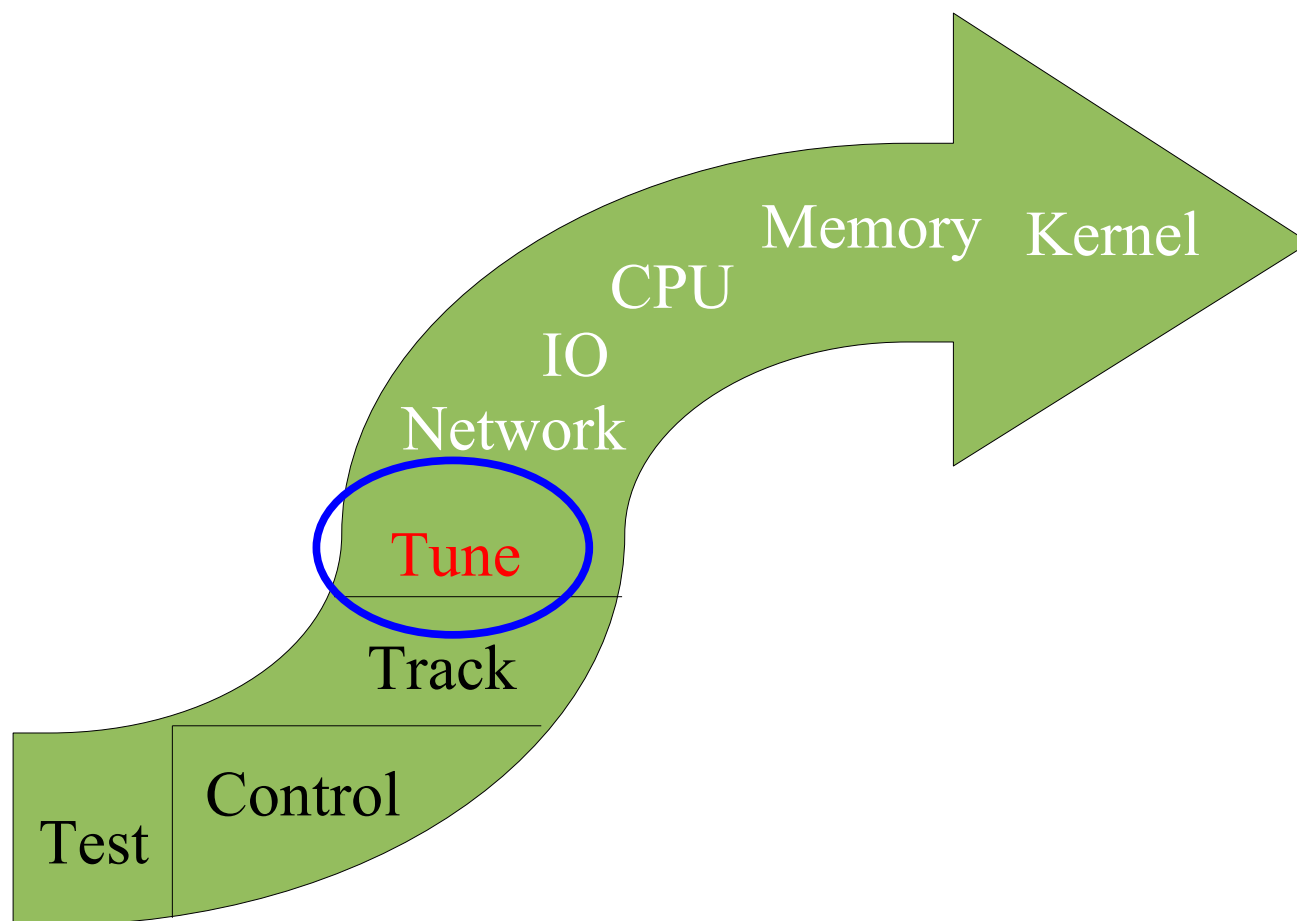




Track

- ✓ syscall エラーの追跡
 - ✓ `dtrace -qn 'syscall:::return/errno!=0 && !progenyof($pid)/{@Errs[execname,probefunc,errno]=count(); } END{printa(@); }'`
 - ✓ ldd (library の確認) ライブラリリンクの確認
 - ✓ `ptree(1)`, `pgrep(1)`, `pfiles(1)` によるアプリケーションの挙動の確認
 - ✓ コマンドプロセスの追跡
 - ✓ `dtrace -Fn 'pid$target:::entry,pid$target:::return{' -c コマンド`
 - ✓ `apptrace(1)`

アプリケーション分析の流れ





Tune

- ✓ Test→Control→Track を通じて、得られた結果を基に必要なチューニングを実施
 - ✓ Test フェーズでは、Network、IO、CPU、Memory に関して基礎値を取得
 - ✓ Control フェーズで、アプリケーション動作を把握
 - ✓ Track フェーズで、アプリケーションのバイナリやエラーなど動作が正しいか検証

Network

- ✓ パフォーマンスツール (Control)
 - ✓ DTrace
 - ✓ IP Provider 利用 (snv93)
 - ✓ mib Provider 利用
 - ✓ netstat(1M)
 - ✓ kstat(1M)
 - ✓ nicstat (K9Toolkit)
 - ✓ intrstat(1M) CPU 割り込みの確認
- ✓ TCP driver のチューニング
 - ✓ 想定しているセッション数を考慮してチューニング
 - ✓ メリットとデメリットは理解する必要あり。
 - ✓ `tcp_[recv_xmit]_hiwat`
 - ✓ `tcp_conn_req_max_q[0]`

netstat(1M)

```
#netstat -af inet
TCP: IPv4
```

Local Address	Remote Address	Swind	Send-Q	Rwind	Recv-Q	State
.	*.*	0	0	49152	0	IDLE
kai01.telnet	fw.37079	49640	0	49640	0	ESTABLISHED
kai01.telnet	fw.37187	49640	0	49640	0	ESTABLISHED
kai01.telnet	fw.37080	48180	0	49640	0	ESTABLISHED
*.sunrpc	*.*	0	0	49152	0	LISTEN
.	*.*	0	0	49152	0	IDLE
*.32771	*.*	0	0	49152	0	LISTEN
*.32772	*.*	0	0	49152	0	LISTEN
*.lockd	*.*	0	0	49152	0	LISTEN
localhost.5987	*.*	0	0	49152	0	LISTEN
localhost.898	*.*	0	0	49152	0	LISTEN
localhost.32773	*.*	0	0	49152	0	LISTEN
localhost.5988	*.*	0	0	49152	0	LISTEN
localhost.32774	*.*	0	0	49152	0	LISTEN

```
#netstat -I ce0 1
```

input ce0		output			input (Total)		output		
packets	errs	packets	errs	colls	packets	errs	packets	errs	colls
2308160	0	2427504	0	0	2370366	0	2489710	0	0
10	0	17	0	0	10	0	17	0	0
11	0	9	0	0	11	0	9	0	0
10	0	16	0	0	10	0	16	0	0

intrstat(1M)

CPU の割り込みを確認することができる

```
#intrstat 1
```

device	cpu0 %tim		cpu1 %tim		cpu2 %tim		cpu3 %tim	
ce#0	0	0.0	0	0.0	6	0.0	0	0.0
mpt#0	0	0.0	0	0.0	0	0.0	0	0.0
ohci#0	0	0.0	0	0.0	0	0.0	0	0.0
ohci#1	0	0.0	0	0.0	0	0.0	0	0.0
qfe#0	0	0.0	0	0.0	0	0.0	8648	12.2
qlc#0	0	0.0	0	0.0	0	0.0	0	0.0
rmc_comm#0	0	0.0	0	0.0	0	0.0	0	0.0
su#0	0	0.0	0	0.0	0	0.0	0	0.0
uata#0	1	0.0	0	0.0	0	0.0	0	0.0

nicstat(K9 Toolkit)

network interface ごとに帯域を計測

```
# nicstat 1
```

Time	Int	rKb/s	wKb/s	rPk/s	wPk/s	rAvs	wAvs	%Util	Sat
12:33:04	hme0	1.51	4.84	7.26	10.32	213.03	480.04	0.05	0.00
12:33:05	hme0	0.20	0.26	3.00	3.00	68.67	90.00	0.00	0.00
12:33:06	hme0	0.14	0.26	2.00	3.00	73.00	90.00	0.00	0.00
12:33:07	hme0	0.14	0.52	2.00	6.00	73.00	88.00	0.01	0.00
12:33:08	hme0	0.24	0.36	3.00	4.00	81.33	92.00	0.00	0.00
12:33:09	hme0	2.20	1.77	16.00	18.00	140.62	100.72	0.03	0.00
12:33:10	hme0	0.49	0.58	8.00	9.00	63.25	66.00	0.01	0.00
12:33:11	hme0	12.16	1830.38	185.06	1326.42	67.26	1413.06	15.09	0.00
12:33:12	hme0	19.03	3094.19	292.88	2229.11	66.53	1421.40	25.50	0.00
12:33:13	hme0	19.55	3151.87	301.00	2270.98	66.50	1421.20	25.98	0.00
12:33:14	hme0	11.99	1471.67	161.07	1081.45	76.25	1393.49	12.15	0.00
12:33:15	hme0	0.14	0.26	2.00	3.00	73.00	90.00	0.00	0.00
12:33:16	hme0	0.14	0.26	2.00	3.00	73.00	90.00	0.00	0.00
12:33:17	hme0	0.14	0.26	2.00	3.00	73.00	90.00	0.00	0.00
[...]									

参照 : <http://www.brendangregg.com/k9toolkit.html>

dtrace(1M)

```
# dtrace -n 'mib:::tcp*{@[tid,execname]=count();} tick-1sec{printa(@);trunc(@);}'
dtrace: description 'mib:::tcp*' matched 95 probes
CPU      ID      FUNCTION:NAME
 3    47398      :tick-1sec
      1    dtrace                2
      1    uperf                 2
      2    uperf                216
      3    uperf                216
      8    uperf                216
      9    uperf                216
     10    uperf                216
     11    uperf                216
      7    uperf                217
      5    uperf                229
      6    uperf                229
      4    uperf                241
      0    sched               19700
```

プロセス、スレッド単位で tcp 通信を確認



IO

- ✓ パフォーマンスツール (Control)
 - ✓ iostat(1M)
 - ✓ fsstat(1M)
 - ✓ kstat(1M)
 - ✓ dtrace(1M)
 - ✓ io provider
 - ✓ fsinfo provider
- ✓ disk IO block size の把握
- ✓ IO アクセスパターンの把握
- ✓ スレッド数の把握
- ✓ IO library の確認
- ✓ アプリケーション IO の確認
- ✓ 物理構成の検討
 - ✓ RAID 構成の検討 (Storage)
 - ✓ ファイルシステム
 - ✓ raw デバイス

iostat(1M)

```
# iostat -xzn 1
                extended device statistics
  r/s    w/s    kr/s    kw/s  wait  actv  wsvc_t  asvc_t   %w   %b  device
  0.7    1.2    6.4    13.7  0.0   0.0   0.0    12.5    0    1  c2t0d0
  0.0    0.0    0.0    0.0   0.0   0.0   0.0     0.9    0    0  c1t0d0
  0.0    0.0    0.0    3.4   0.0   0.0  25.6    98.0    0    0  c0t44d0
```

fsstat(1M)

```
# fsstat -F 1
new name      name attr attr lookup rddir  read read  write write
file remov   chng  get  set  ops  ops  ops bytes ops bytes
3.33K 2.00K    874 259K 1.05K 856K 5.74K 96.1K 175M 4.25K 15.4M ufs
  0    0      0 4.06K 0    7.39K 200 1.62K 372K 0    0 proc
  0    0      0 3    0    0    0 0    0 0    0 nfs
  0    0      0 0    0    0    0 0    0 0    0 zfs
  0    0      0 10.5K 0    0    0 0    0 0    0 lofs
5.87K 3.80K    680 36.8K 80 15.3K 2 90.8K 93.4M 62.0K 64.0M tmpfs
  0    0      0 56 0    0    0 27 3.31K 0    0 mntfs
  0    0      0 0 0    0    0 0 0    0 0    0 nfs3
  0    0      0 17.7K 0 237K 1.15K 5.76K 331K 0    0 nfs4
  0    0      0 121K 0 121K 0 0 0 0    0 autofs
```



dtrace(1M)

物理 IO の確認

```
# dtrace -n 'io:::start{@[execname,args[2]->fi_pathname]=quantize(args[0]->b_bcount);}'
dtrace: description 'io:::start' matched 6 probes
```

```
# ../bin/filebench io
parsing profile for config: randomread2k
Running /tmp/sagami08.jp.iforce.net-tmpfs-io-Jun_28_2008-04h_51m_59s/randomread2k/thisrun.f
FileBench Version 1.3.3
28861: 0.031: Random Read Version 2.0 IO personality successfully loaded
28861: 0.032: Creating/pre-allocating files and filesets
28861: 0.033: File largefile1: mbytes=10240
28861: 0.033: Creating file largefile1...
28861: 0.034: Preallocated 1 of 1 of file largefile1 in 1 seconds
28861: 0.034: waiting for fileset pre-allocation to finish
28861: 179.507: Running '/opt/filebench/scripts/fs_flush tmpfs /tmp'
filesystem type is: tmpfs, no action required, so exiting
28861: 179.528: Change dir to /tmp/sagami08.jp.iforce.net-tmpfs-io-Jun_28_2008-04h_51m_59s/randomread2k
28861: 179.528: Starting 1 rand-read instances
28872: 180.532: Starting 16 rand-thread threads
28861: 183.533: Running...
    go_filebench
    /fs/Logfile/00000001/00000
    001
        value  ----- Distribution -----  count
        4096  |
        8192  | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 64
        16384 |
    fsflush
    /var/tmp/fbenchIPaGmh
        value  ----- Distribution -----  count
        4096  |
        8192  | @@@@@@ 5
        16384 | @@@@@@@@@@@@@@@@ 9
        32768 | @@@@@@@@@@@@@@ 8
        65536 | @@@@@@@@@@@@@@ 9
        131072| 0
```



dtrace(1M)

どんなアプリケーションが IO をしているか？

```
#!/usr/sbin/dtrace -s
io:::start {
    @[execname,args[2]->fi_pathname] = count();
}
```

IO しているアプリケーションのブロックサイズは？

```
#!/usr/sbin/dtrace -s
io:::start {
    @[execname,args[1]->dev_statname]=quantize(args[0]->b_bcount);
}
```

dtrace(1M)

filesystem への IO の確認

```
# dtrace -n 'fsinfo:::{@[args[0]->fi_fs,probename]=count();} tick-1sec {printa(@);trunc(@)}'
```

```
# ../bin/filebench io
parsing profile for config: randomread2k
Running /tmp/sagami08.jp.iforce.net-tmpfs-io-Jun_28_2008-04h_51m_59s/randomread2k/thisrun.f
FileBench Version 1.3.3
28861: 0.031: Random Read Version 2.0 IO personality successfully loaded
28861: 0.032: Creating/pre-allocating files and filesets
28861: 0.033: File largefile1: mbytes=10240
28861: 0.033: Creating file largefile1...
28861: 0.034: Preallocated 1 of 1 of file largefile1 in 1 seconds
28861: 0.034: waiting for fileset pre-allocation to finish
28861: 179.507: Running '/opt/filebench/scripts/fs_flush tmpfs /tmp'
filesystem type is: tmpfs, no action required, so exiting
28861: 179.528: Change dir to /tmp/sagami08.jp.iforce.net-tmpfs-io-
Jun_28_2008-04h_51m_59s/randomread2k
28861: 179.528: Starting 1 rand-read instances
28872: 180.532: Starting 16 rand-thread threads
28861: 183.533: Running...
```

3	51722	:tick-1sec		
ufs			putpage	1
tmpfs			getpage	1479
tmpfs			read	1817
tmpfs			rwlock	1817
tmpfs			rwunlock	1817
proc			read	3634
proc			rwlock	3634
proc			rwunlock	3634
tmpfs			putpage	60913

dtrace(1M)

どんなアプリケーションが論理 IO を発行しているか？

```
#!/usr/sbin/dtrace -s
fsinfo:::read,fsinfo:::write {
    @[execname,pid,tid,args[0]->fi_fs]=count();
}
```

どんなアプリケーションがどんな論理 IO サイズを発行しているか？

```
#!/usr/sbin/dtrace -s
fsinfo:::read,fsinfo:::write {
    @[execname,pid,tid,args[0]->fi_fs]=quantize(args[1]);
}
```

IO アクセスパターン

- ✓ DTrace ToolKit (iopattern)

```
# ./iopattern
%RAN %SEQ COUNT MIN MAX AVG KR KW
0 100 105 131072 131072 131072 0 13440
0 100 106 131072 131072 131072 0 13568
0 100 108 131072 131072 131072 0 13824
0 100 109 131072 131072 131072 0 13952
0 100 109 131072 131072 131072 0 13952
```

^C

ディスクアクセスが sequential/random かその割合、IO サイズを把握できる

<http://www.opensolaris.org/os/community/dtrace/dtracetoolkit>

CPU

- ✓ パフォーマンスツール (Control)
 - ✓ cputrack(1)
 - ✓ cpustat(1M)
 - ✓ prstat -mL
 - ✓ dtrace(1M)
 - ✓ profile プロバイダ
 - ✓ sched プロバイダ
 - ✓ mpstat(1M)
 - ✓ plockstat(1M)
- ✓ スケジューラの変更
 - ✓ priocntl(1) で調整
 - ✓ FSS
 - ✓ Fixed priority
- ✓ CPU partition
 - ✓ スレッド移動の回避
 - ✓ プロセッサセットの利用
 - ✓ プロセスバインド
- ✓ plockstat にて、ユーザーレベル
ロックの確認

cputrack(1)

- ✓ CPC HW counter を利用して、アプリケーションのキャッシュミス等を検出

```
# cputrack -c "pic0=Instr_cnt,pic1=DTLB_miss" 1s
  time lwp      event      pic0      pic1
atom.c
 0.009   1      exit  294869   92
```

mpstat(1M)

マルチプロセッサの統計値の確認

```
# mpstat 1
CPU minf mjf xcal  intr  ithr  csw  icsw  migr  smtx  srw  syscl  usr  sys  wt  idl
0    21   0   31   16    7   77    5   10   12    1   267    1    1    0   99
1    12   0   16   17    5   71    5    9   12    1   166    0    0    0   99
2    11   0    9   23   11  101    5    9   12    1   181    1    0    0   99
3    24   0   15  354  242  178    5   11   13    1    75    0    2    0   97
```

lockstat(1M)

- ✓ カーネルのプロファイル
- ✓ ロックイベントの競合情報 (mutex lock, rw lock, thread lock etc..)
- ✓ mpstat(1M) の値をみて取得

```
# lockstat -i 997 -C sleep 5
Adaptive mutex spin: 2 events in 5.053 seconds (0 events/sec)
Count indiv cuml rcnt      spin Lock                      Caller
-----
   1  50%  50% 0.00          1 0x6000172bd00          anon_resvmem+0x34
   1  50% 100% 0.00          1 0x30000c7b290          ufs_lockfs_end+0x70
-----
Adaptive mutex block: 1 events in 5.053 seconds (0 events/sec)
Count indiv cuml rcnt      nsec Lock                      Caller
-----
   1 100% 100% 0.00 102297500 0x60001860018          mpt_ioctl+0x1c8
...

```

prstat(1)

プロセス・スレッドの CPU 利用率の確認
CPU 待ちをしているスレッドの確認

```
#prstat -mL 1
PID USERNAME      USR  SYS  TRP  TFL  DFL  LCK  SLP  LAT  VCX  ICX  SCL  SIG  PROCESS/LWPID
28308 root            0.1  12  0.0  0.0  0.0  0.0  88  0.0  53   2  265   0  go_filebench/8
28308 root            0.1  11  0.0  0.0  0.0  0.0  89  0.0  54   2  270   0  go_filebench/6
28308 root            0.1  8.3  0.0  0.0  0.0  0.0  92  0.0  40   2  200   0  go_filebench/11
28308 root            0.1  8.2  0.0  0.0  0.0  0.0  92  0.0  35   1  175   0  go_filebench/13
28308 root            0.1  7.9  0.0  0.0  0.0  0.0  92  0.0  41   2  205   0  go_filebench/12
28308 root            0.1  7.1  0.0  0.0  0.0  0.0  93  0.0  32   3  160   0  go_filebench/14
28308 root            0.1  7.0  0.0  0.0  0.0  0.0  93  0.0  35   1  175   0  go_filebench/10
28308 root            0.0  6.8  0.0  0.0  0.0  0.0  93  0.0  30   1  150   0  go_filebench/17
28308 root            0.0  6.2  0.0  0.0  0.0  0.0  94  0.0  31   1  155   0  go_filebench/2
28308 root            0.0  6.2  0.0  0.0  0.0  0.0  94  0.0  27   1  135   0  go_filebench/3
28308 root            0.0  5.3  0.0  0.0  0.0  0.0  95  0.0  23   1  115   0  go_filebench/5
28308 root            0.0  4.1  0.0  0.0  0.0  0.0  96  0.0  19   1   95   0  go_filebench/4
28308 root            0.0  3.6  0.0  0.0  0.0  0.0  96  0.0  16   1   80   0  go_filebench/16
28308 root            0.0  3.2  0.0  0.0  0.0  0.0  97  0.0  16   2   80   0  go_filebench/9
28308 root            0.0  2.4  0.0  0.0  0.0  0.0  98  0.0  12   1   60   0  go_filebench/15
28308 root            0.0  2.3  0.0  0.0  0.0  0.0  98  0.0  11   2   55   0  go_filebench/7
28309 root            0.3  1.1  0.1  0.0  0.0  0.0  99  0.0  23   0  346   0  prstat/1
  158 root            0.0  0.0  0.0  0.0  0.0  0.0 100  0.0  21   0   86   0  nscd/15
  532 root            0.0  0.0  0.0  0.0  0.0  0.0 100  0.0   3   0    2   0  httpd/1
28259 root            0.0  0.0  0.0  0.0  0.0  0.0 100  0.0   3   0    1   0  go_filebench/2
28259 root            0.0  0.0  0.0  0.0  0.0  0.0 100  0.0   3   0    1   0  go_filebench/1
Total: 94 processes, 234 lwps, load averages: 0.90, 0.36, 0.22
```



profile provider(usr)

CPU%(usr) で多く利用されている関数を追跡

```
#dtrace -n 'profile-997/arg1&&!progenyof($pid)/{@[execname,ufunc(arg1)]=count();}'
dtrace: description 'profile-997' matched 1 probe
c^C
```

snmpd	libc.so.1`gettimeofday	1
ls	0xff3be468	2
snmpd	libkstat.so.1`kstat_chain_update	2

profile provider(sys)

CPU%(sys) で多く利用されている関数を追跡

```
# dtrace -n 'profile-997/arg0&&!progenyof($pid)/{
@[execname,func(arg0),curthread->t_startpc]=count();}
END{printa("\n %s %a %a %d\n",@);}'
dtrace: description 'profile-997' matched 2 probes
^C
CPU   ID          FUNCTION:NAME
0     2             :END
fsflush unix`mutex_exit 0x0 1
go_filebench unix`hat_getpfnum 0x0 1
go_filebench unix`lock_clear 0x0 1
go_filebench tmpfs`tmp_putpage 0x0 1
...
sched unix`i_ddi_splhigh unix`idle 805
sched unix`i_ddi_splx unix`idle 1022
sched unix`idle unix`idle 1271
sched unix`disp_getwork unix`idle 4911
```



Memory

- ✓ パフォーマンスツール (Control)
 - ✓ vmstat(1M) 仮想メモリ統計
 - ✓ dtrace(1M)
 - ✓ vminfo プロバイダ
 - ✓ prstat -mL プロセス統計
- ✓ メモリのチューニング
 - ✓ メモリアロケータ
 - ✓ Large Page Size
 - ✓ MPO(SMP の場合考慮)

メモリアロケータ

- ✓ プロセスごとに、pmap(1)にて mapfile の確認
- ✓ libc.so.1 メモリアロケータに注意
- ✓ メモリアロケータを変更する
 - ✓ たとえば、libumem(3LIB) を利用することも考慮
 - ✓ LD_PRELOAD=\$LD_PRELOAD:libumem.so.1
 - ✓ export LD_PRELOAD



Large Page Size

- ✓ LD_PRELOAD=
\$LD_PRELOAD:mpss.so.1
- ✓ pmap -xs にて、変更されたかどうかを確認できる。
- ✓ MPSSHEAP=size
 - ✓ プロセスヒープの
pagesize 設定
- ✓ MPSSSTACK=size
 - ✓ プロセススタックの
pagesize 設定

vmstat(1M)

```
#vmstat 1
kthr      memory          page        disk        faults        cpu
 r  b  w    swap  free  re  mf  pi  po  fr  de  sr  s1  sd  sd  --   in   sy   cs  us  sy  id
 4  0  0 30289776 29324440 15 100 21 12 12  0  1 -0 -1  3  0  423  725  510  0  0  99
 0  0  0 30279160 29211352 86 108  0  0  0  0  0  0  0  0  0  372  649  224  0  0 100
 0  0  0 30279136 29211384  0  3  0  0  0  0  0  0  0  0  0  350  121  218  0  0 100
 0  0  0 30279136 29211384  0  0  0  0  0  0  0  0  0  0  0  448  211  259  0  0 100
 0  0  0 30279136 29211384  0  0  0  0  0  0  0  0  0  0  0  348  109  218  0  0 100
```

```
#vmstat -p 1
      memory          page        executable      anonymous      filesystem
      swap  free  re  mf  fr  de  sr  epi  epo  epf  api  apo  apf  fpi  fpo  fpf
30289800 29324616 15 100 12  0  1    1    0    0    0    0    0    20   12   12
30279192 29211384  7 19  0    0  0    0    0    0    0    0    0    0    0    0
30279168 29211416  0 3  0    0  0    0    0    0    0    0    0    0    0    0
```



pmap(1)

```
# pmap -xs `pgrep malloc`
6370: bin/./bin-sun4u/malloc
Address  Kbytes  RSS    Anon  Locked Pgsz  Mode  Mapped File
00010000    24     24     -     -     8K  r-x--  malloc
00024000     8      8      8     -     8K  rwx--  malloc
00026000    16     16     16    -     8K  rwx--  [ heap ]
FF080000     8      8     -     -     8K  rw-s-  [ anon ]
FF082000   776     -     -     -     -  rw-s-  [ anon ]
FF180000   200    200     -     -     8K  r-x--  libc.so.1
FF1B2000    56     56     -     -     -  r-x--  libc.so.1
FF1C0000    64     64     -     -     8K  r-x--  libc.so.1
FF1D0000    16     16     -     -     -  r-x--  libc.so.1
FF1D4000    64     64     -     -     8K  r-x--  libc.so.1
FF1E4000     8      8     -     -     -  r-x--  libc.so.1
FF1E6000    56     56     -     -     8K  r-x--  libc.so.1
FF1F4000   208    208     -     -     -  r-x--  libc.so.1
FF228000   136    136     -     -     8K  r-x--  libc.so.1
...
```

plockstat(1M)

- ✓ アプリケーションの競合を検出することができる。
- ✓ prstat -mL の LCK が高い時に取得すると lock の競合を検出

```
# plockstat -x bufsize=20m -A malloc -T 3
^C
Mutex hold
```

Count	nsec	Lock	Caller
1	113087400	0xff3735c0	0xff337dc4
3	7583066	0xff3735c0	0xff337dc4
3	7563100	0xff374540	libm.so.2`gam_n+0x51b0
3	7539700	0xff373318	libm.so.2`atanl+0x17ec
3	7491800	0xff371838	libm.so.2`atanl+0x17e4
1	6484500	libc.so.1`_uberdata+0x40	malloc`actual_main+0x710
1	6471800	libc.so.1`_uberdata+0xfc0	malloc`actual_main+0x710
1	6449700	libc.so.1`_first_link_lock	malloc`actual_main+0x710
1	6432800	libc.so.1`libc_malloc_lock	malloc`actual_main+0x710
1	107300	libc.so.1`libc_malloc_lock	malloc`crunch_stats+0xc8
...			

DProfile

- ✓ SunStudio11 から導入されたハードウェアプロファイリングの機能
- ✓ アプリケーションは、プロセスから構成
- ✓ プロセスは、スレッドから構成
- ✓ スレッドは、strand 上でスケジュールされている
- ✓ シンボルがストリップされていなければ、さらに深くみる
ことができる。
 - ✓ スレッドは、instructions から構成
 - ✓ インストラクションは、関数のグループに所属
 - ✓ 関数は、ロードオブジェクトにグループされる。
- ✓ DProfile(dataspace profile) を利用すると、
 - ✓ CPU のどのキャッシュラインを利用されているかを理解
できる
 - ✓ ハードウェアからみるキャッシュラインの利用の仕方を
みて、最適な多重度を調整する(ハードウェアの観点で)

パフォーマンスシナリオ

スレッド処理で演算処理を行うプログラム

```
# ptime ./atom 100000
```

```
real    19.228
user    39.056
sys     25.742
```

```
void consumer(void *foo)
{
    while(mytimes < (COUNT - 1)) {
        mutex_lock(&thelock);
        mytimes++;
        mutex_unlock(&thelock);
    }
}
```



Test フェーズは、終了
Control フェーズ

メモリの使用状況は？

```
# vmstat -p 1
      memory
      swap  free  re  mf  fr  de  sr
30288328 29309088 13 87 11 0   1
30277656 29209976 7 19 0  0   0
30277632 29210008 0 3  0  0   0
      page
      executable
      anonymous
      filesystem
      epi  epo  epf  api  apo  apf  fpi  fpo  fpf
1       0   0   0   0   0   17  11  11
0       0   0   0   0   0   0   0   0
0       0   0   0   0   0   0   0   0
```

prstat でわかるでしょうか？

```
# prstat
PID USERNAME  SIZE  RSS STATE  PRI NICE  TIME  CPU PROCESS/NLWP
7707 root      75M  75M  cpu19  20  0    0:00:46 2.7% atom/59410
7706 root     3448K 3096K  cpu1   59  0    0:00:00 0.0% prstat/1
3602 root     150M  75M  sleep  59  0    0:00:31 0.0% java/25
678 root     8248K 6544K  sleep  59  0    0:00:40 0.0% picld/11
13015 root    2392K 1608K  sleep  59  0    0:00:00 0.0% ttymon/1
```

仮説 1 割り込み (intrstat)

```
# intrstat 5
```

device	cpu0 %tim	cpu1 %tim	cpu2 %tim	cpu3 %tim
ce#0	0 0.0	0 0.0	0 0.0	0 0.0
uata#0	4 0.0	0 0.0	0 0.0	0 0.0

device	cpu16 %tim	cpu17 %tim	cpu18 %tim	cpu19 %tim
ce#0	0 0.0	0 0.0	0 0.0	2 0.0
uata#0	0 0.0	0 0.0	0 0.0	0 0.0

仮説 2 TLB/TSB ミス

```
#trapstat -T 1
```

cpu	m	size	itlb-miss %tim	itsb-miss %tim	dtlb-miss %tim	dtsb-miss %tim	%tim
0	u	8k	75 0.0	0 0.0	2046 0.1	2014 0.1	0.2
0	u	64k	0 0.0	0 0.0	0 0.0	0 0.0	0.0
0	u	512k	0 0.0	0 0.0	0 0.0	0 0.0	0.0
0	u	4m	0 0.0	0 0.0	1 0.0	0 0.0	0.0
0	u	32m	0 0.0	0 0.0	0 0.0	0 0.0	0.0
0	u	256m	0 0.0	0 0.0	0 0.0	0 0.0	0.0
0	k	8k	3 0.0	0 0.0	16805 0.6	1530 0.1	0.7
0	k	64k	0 0.0	0 0.0	0 0.0	0 0.0	0.0
0	k	512k	0 0.0	0 0.0	0 0.0	0 0.0	0.0
0	k	4m	0 0.0	0 0.0	496 0.0	0 0.0	0.0
0	k	32m	0 0.0	0 0.0	0 0.0	0 0.0	0.0
0	k	256m	0 0.0	0 0.0	0 0.0	0 0.0	0.0

仮説3 短期間のプロセス

DTraceToolkit execsnoop

どんなアプリケーションが実行されたか。

```
# execsnoop -v
STRTIME          UID      PID      PPID  ARGS
2008 Jul  1 14:48:14    0      7765    7728 ./atom 10000
```

DTraceToolKit shortlived.d

どんなプロセスが常駐していたか。

```
# shortlived.d
Tracing... Hit Ctrl-C to stop.
^C
short lived processes:      0.402 secs
total sample duration:    20.378 secs
```

Total time by process name,

ls	3 ms
atom	397 ms

アプリケーションはマルチスレッドか？

```
# ptime ./atom 10
```

```
real    4.659
user    25.522
sys     0.428
```

```
# ptime ./atom 100000
```

```
real    14.293
user    37.961
sys     21.776
```

```
# prstat -mL 1
```

PID	USERNAME	USR	SYS	TRP	TFL	DFL	LCK	SLP	LAT	VCX	ICX	SCL	SIG	PROCESS/LWPID
3661	root	1.5	9.9	0.0	0.0	0.0	0.0	0.6	88	1	31	4K	0	atom/1
3659	root	2.3	1.1	0.0	0.0	0.0	0.0	32	65	24	63	466	0	prstat/1
3661	root	3.0	0.0	0.0	0.0	0.0	8.0	0.0	89	16	5	22	0	atom/1490
3661	root	2.5	0.4	0.0	0.0	0.0	22	0.0	75	17	0	100	0	atom/773
3661	root	2.6	0.2	0.0	0.0	0.0	42	0.0	56	19	0	51	0	atom/492
3661	root	2.6	0.1	0.0	0.0	0.0	66	0.0	31	32	0	63	0	atom/18
3661	root	2.7	0.0	0.0	0.0	0.0	18	0.0	80	13	0	13	0	atom/1260
3661	root	2.4	0.3	0.0	0.0	0.0	24	0.0	73	14	1	89	0	atom/347
3661	root	2.6	0.1	0.0	0.0	0.0	26	0.0	72	14	0	20	0	atom/1009
3661	root	2.6	0.0	0.0	0.0	0.0	11	0.0	86	17	1	29	0	atom/615
3661	root	2.0	0.6	0.0	0.0	0.0	64	0.0	33	28	1	176	0	atom/49
3661	root	2.5	0.0	0.0	0.0	0.0	23	0.0	74	19	0	22	0	atom/1329
3661	root	2.4	0.1	0.0	0.0	0.0	6.8	0.0	91	13	1	35	0	atom/1210
3661	root	2.5	0.0	0.0	0.0	0.0	24	0.0	74	12	0	15	0	atom/916
3661	root	2.5	0.0	0.0	0.0	0.0	56	0.0	41	22	0	28	0	atom/291
3661	root	2.4	0.0	0.0	0.0	0.0	25	0.0	72	17	2	21	0	atom/731
3661	root	2.4	0.0	0.0	0.0	0.0	27	0.0	70	25	4	34	0	atom/327
3661	root	2.4	0.0	0.0	0.0	0.0	25	0.0	72	20	1	25	0	atom/268
3661	root	2.4	0.1	0.0	0.0	0.0	24	0.0	74	17	1	39	0	atom/1021

```
# plockstat -p `pgrep atom`
Mutex block
```

ロックの競合は？

Count	nsec	Lock	Caller
1627	505751874	libc.so.1`_ubedata	libc.so.1`_thr_exit_common+0xbc
1910	387812755	libc.so.1`_ubedata	libc.so.1`_thr_exit_common+0xbc
220	353002605	atom`thelock	atom`consumer+0x3c
10	159899890	libc.so.1`_ubedata	atom`main+0x88
110	8850657	atom`thelock	atom`consumer+0x3c
87	271847	libc.so.1`_ubedata	libc.so.1`_thr_exit_common+0xbc
949	24591	libc.so.1`_ubedata	libc.so.1`_thr_exit_common+0xbc
1	2981500	libc.so.1`_ubedata	atom`main+0x6c
1	5000	libc.so.1`_ubedata	atom`main+0x88

```
# LD_PRELOAD=$LD_PRELOAD:libumem.so.1
```

```
# export LD_PRELOAD
```

```
# ptime ./atom 100000
```

```
# plockstat -p `pgrep atom`
plockstat: pid 3745 has exited
```

メモリアロケータの変更

```
real    6.551
user    37.106
sys     11.804
```

```
Mutex block
```

Count	nsec	Lock	Caller
16	2628687	libc.so.1`_ubedata	atom`main+0x4c
244	4689	libc.so.1`_ubedata	atom`main+0x4c
141	5050	libc.so.1`_ubedata	libc.so.1`thr_create+0x2c
11	51381	libc.so.1`_ubedata	libc.so.1`_thr_exit_common+0xbc
23	20717	libc.so.1`_ubedata	libc.so.1`_thr_exit_common+0xbc
10	29380	libc.so.1`_ubedata	libc.so.1`_thr_exit_common+0xbc
3	60733	libc.so.1`_ubedata	libc.so.1`thr_create+0x2c
3	51200	libc.so.1`_ubedata	libc.so.1`_thr_exit_common+0xbc

DProfile

- バイナリコンパイル時に、`-xhwcprof` オプションをつける。
`# cc atom.c -fast -g -xhwcprof -o atom`
- `psig` で、プログラムのシグナル配置を確認する。（必要ならば）
- データ集計する
- `.er.rc` のオブジェクト定義を確認（参考ページ）
- `analyzer` を起動

The screenshot shows the Sun Studio Analyzer interface. The main window displays a table of functions with columns for User CPU, Max. Mem. Stall, and Name. The function `take_deferred_signal` is highlighted, showing a User CPU of 14.700 and Max. Mem. Stall of 0. The right-hand pane shows the 'Selected Object' details for `take_deferred_signal`, including its PC Address, Size, Source File, Object File, and Load Object. Below this, a 'Metrics for Selected Object' table provides a breakdown of CPU and memory usage, distinguishing between Exclusive and Inclusive metrics.

	Exclusive	Inclusive
Max. Mem. Stall:	0. (0.%)	0. (0.%)
User CPU:	14.700 (82.20%)	14.700 (82.20%)
Wall:	4.293 (67.99%)	4.293 (67.99%)
Total LWP:	1856.088 (14.17%)	1856.088 (14.17%)
System CPU:	9.647 (41.36%)	9.647 (41.36%)
Wait CPU:	1509.566 (18.91%)	1509.566 (18.91%)
User Lock:	321.415 (6.33%)	321.415 (6.33%)
Text Page Fault:	0. (0.%)	0. (0.%)
Data Page Fault:	0. (0.%)	0. (0.%)

DProfile の利用

- 正確にロードされると、プロファイルデータから cache line の表示
- どのキャッシュラインが、時間がかかるかを把握

Max. Mem. Stall (sec.)	Name
1.011	<Total>
0.510	US4p_L2CacheLine Memory Object 7209
0.360	US4p_L2CacheLine Memory Object 7208
0.070	US4p_L2CacheLine Memory Object 3870
0.010	US4p_L2CacheLine Memory Object 0
0.010	US4p_L2CacheLine Memory Object 42
0.010	US4p_L2CacheLine Memory Object 138
0.010	US4p_L2CacheLine Memory Object 203
0.010	US4p_L2CacheLine Memory Object 971
0.010	US4p_L2CacheLine Memory Object 6635
0.010	US4p_L2CacheLine Memory Object 8043
0.	US4p_L2CacheLine Memory Object 3402
0.	US4p_L2CacheLine Memory Object 5631

Selected Object: US4p_L2CacheLine Memory Object 7209

Metrics for Selected Object:

Max. Mem. Stall: 0.510 (50.50%)



DProfile の結果を受けて、スレッド数の変更を行う

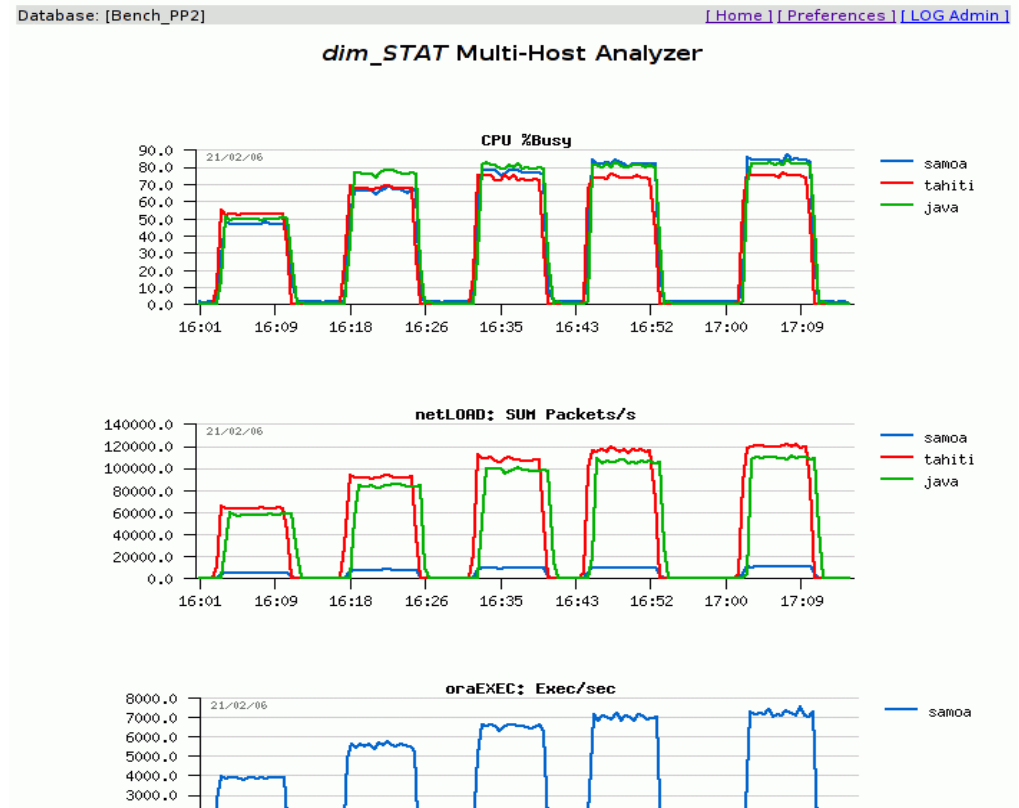


デモンストレーション

- ✓ libmicro の malloc 負荷を利用して、下記の効果を見る。
 - ✓ libumem.so.1
- ✓ その際に利用する、パフォーマンスツールは、
 - ✓ prstat -mL
 - ✓ plockstat
- ✓ ハードウェア構成
 - ✓ V490
 - ✓ Ultra SPARC IV+ 1.5GHz x 4 (dual core)
 - ✓ Memory 32GB
 - ✓ Disk 146GB(15krpm)

ベンチマークデータ解析ツール

- ✓ dimstat 8.1
- ✓ 主な特徴
 - ✓ システム統計情報を MySQL に格納
 - ✓ レポート作成ツール
 - ✓ システム統計取得ツール
 - ✓ <http://dimitrik.free.fr/>



システムパフォーマンスで考慮する点

- ✓ CPU
 - ✓ スケジューラ
 - ✓ プロセス / スレッド
 - ✓ アプリケーションロック競合
 - ✓ カーネルロック競合
- ✓ Memory
 - ✓ 共有メモリ、プロセスヒープ
 - ✓ メモリアロケータ
 - ✓ Java メモリ管理 (JVM)
 - ✓ ラッチ競合
- ✓ Disk
 - ✓ ファイルシステム (ufs, vxfs, zfs, qfs etc..) , raw デバイス
 - ✓ I/O パターン
 - ✓ ファイルアクセス競合 (スレッド数)
 - ✓ ブロックサイズ
 - ✓ RAID 構成

システムパフォーマンスで考慮する点 (2)

- ✓ どこに負荷がかかる？
 - ✓ Network
 - ✓ Network → ボトルネック → 原因は？帯域の問題？
 - ✓ I/O → ボトルネック → 原因は？ ディスクコントローラを増やす？
 - ✓ Memory → ボトルネック → 原因は？ 容量？
 - ✓ CPU → ボトルネック → 原因は？ CPU の数を増やす？
 - ✓ アプリ (プロセス数 / スレッド数) がどのくらいメモリを利用しているか？
- ✓ CPU スケーラビリティを持たないアプリケーション
 - ✓ シングルスレッドアプリケーション
 - ✓ ロックの競合
 - ✓ I/O ボトルネック

アプリケーションのIO特性

負荷	ファイルサイズ	ファイル数	I/O Mix	Seek Mode	Typical IOPS	IO Size
Web Server	Small	Large	<5% 50r/50w 1% large sequential	Random Rread /10% Sequential Write	<1000 per client	<64k
Small DB	Large	Small	<5% 50r/50w 1%large sequential	99% Random	~1000	Random 2-8k, 128k
Large DB	Large	Small	large seqential	99% Random	>10000	Random 2-8k, 128k
NFS Mail Server	Moderate	Moderate	?	sequential	Low	Large reads, small writes
HPTC	Huge	Small	50r/50w	sequential	~1000?	~1MB
Video Streaming	Small	Large	5r/5w/90a	sequential	~1000	~32k

参考 <http://www.nasconf.com/pres05/mcdougall.pdf>

まとめ

パフォーマンスツール

- ✓ Solaris システムパフォーマンス解析方法の理解
 - ✓ Test → Control → Track → Tune フェーズ
- ✓ システムパフォーマンス解析に必要なツールの紹介

- ✓ Network
 - ✓ dtrace
 - ✓ netstat
 - ✓ nicstat
 - ✓ intrstat
- ✓ IO
 - ✓ iostat
 - ✓ fsstat
 - ✓ dtrace
- ✓ CPU
 - ✓ cputrack
 - ✓ cpustat
 - ✓ prstat -mL
 - ✓ dtrace
 - ✓ mpstat
 - ✓ plockstat
- ✓ Memory
 - ✓ vmstat
 - ✓ dtrace
 - ✓ prstat -mL

Sun Solution Center Tokyo



- ✓ Benchmark/Proof of Concept
- ✓ サンの最新のハードウェアを使って、POC/ベンチマークテストを行うことが可能です。
- ✓ 通常2～3週間ベンチマーク期間
- ✓ 利用する際は、弊社担当営業まで



Gloabl SSC: <http://www.sun.com/solutioncenters/locations/tokyo/>
 日本のページ: <http://jp.sun.com/company/partners/solutioncenter/>



參考資料

- ✓ DTrace
<http://opensolaris.org/os/community/dtrace/>
- ✓ DProfile
<http://blogs.sun.com/nk>
- ✓ MPO(OpenSolaris Performance Community)
<http://opensolaris.org/os/community/performance/>
- ✓ FX Scheduler
http://www.solarisinternals.com/wiki/index.php/FX_For_Databases
- ✓ libumem, mpss
<http://sdc.sun.co.jp/solaris/general/memory.html>
- ✓ uperf
<http://www.uperf.org/>
- ✓ filebench
<http://sourceforge.net/projects/filebench>
- ✓ libmicro
<http://opensolaris.org/os/project/libmicro/>

メモリアロケータ (1)

```
#!/usr/sbin/dtrace -Fs

pid$target::malloc:entry {
    self->ts=1;
}

fbt:::/self->ts/{
}

pid$target::malloc:return/self->ts/{
    exit(0);
}
```

メモリアロケータ
の詳細確認

メモリアロケータ (2) libc

```
[root@sagami08:dtrace]# ./malloc -c
/libmicro/libMicro-0.4.0/bin/malloc libc.so.1
dtrace: script './malloc.d' matched 468 probes
CPU FUNCTION
0 -> malloc          libc.so.1 malloc
0 => brk              brk
0 <= brk             brk
0 => brk              brk
0 <= brk             brk
0 <- malloc          libc.so.1 malloc
```

メモリアロケータ (3) libumem

```
# ./malloc_syscall.d -c /libmicro/libMicro-0.4.0/bin/malloc libumem.so.1
```

```
dtrace: script './malloc_syscall.d' matched 468 probes
```

```
CPU FUNCTION
```

```
3 -> malloc                libumem.so.1 malloc
3 => sysconfig              sysconfig
3 <= sysconfig              sysconfig
3 => issetugid              issetugid
3 <= issetugid              issetugid
3 => issetugid              issetugid
3 <= issetugid              issetugid
3 => brk                    brk
3 <= brk                    brk
3 => brk                    brk
3 <= brk                    brk
3 <- malloc                libumem.so.1 malloc
```

メモリアロケータ関数 (1)

```
#!/usr/sbin/dtrace -Fs

pid$target:$$1:malloc:entry {
    self->ts=1;
    printf("%s %s\n",probemod,probefunc);
}

syscall::/self->ts/ {}

pid$target:$$1::entry,pid$target:$$1::return/self->ts/ {
    printf("%s %s\n",probemod,probefunc);
}

pid$target:$$1:malloc:return/self->ts/ {
    printf("%s %s\n",probemod,probefunc);
    exit(0);
}
```

malloc.d

メモリアロケータ関数 (2)

```
# ./malloc.d -c /libmicro/libMicro-0.4.0/bin/malloc libc.so.1
dtrace: script './malloc.d' matched 5574 probes
```

```
CPU FUNCTION
3 -> malloc                libc.so.1 malloc
3 | malloc:entry          libc.so.1 malloc
3 -> mutex_lock            libc.so.1 mutex_lock
3 <- mutex_lock           libc.so.1 mutex_lock
...
3 -> realloc               libc.so.1 realloc
3 <- realloc               libc.so.1 realloc
3 <- _malloc_unlocked     libc.so.1 _malloc_unlocked
3 -> mutex_unlock         libc.so.1 mutex_unlock
3 <- mutex_unlock         libc.so.1 mutex_unlock
3 | malloc:return         libc.so.1 malloc
```

```
#export LD_PRELOAD=libumem.so.1
# ./malloc.d -c /libmicro/libMicro-0.4.0/bin/malloc libumem.so.1
dtrace: script './malloc.d' matched 5574 probes
```

```
CPU FUNCTION
3 -> malloc                libumem.so.1 malloc
3 | malloc:entry          libumem.so.1 malloc
3 -> umem_alloc            libumem.so.1 umem_alloc
3 -> umem_cache_alloc     libumem.so.1
umem_cache_alloc
3 -> umem_slab_alloc      libumem.so.1
umem_slab_alloc
3 <- umem_slab_alloc      libumem.so.1
umem_slab_alloc
...
3 <- umem_log_event       libumem.so.1
umem_log_event
3 <- umem_slab_alloc      libumem.so.1
umem_slab_alloc
3 <- umem_cache_alloc     libumem.so.1
umem_cache_alloc
3 <- umem_alloc            libumem.so.1 umem_alloc
```

.er.rc の定義

```
# cat .er.rc
en_desc on
ignore_no_xhwcprof
mobj_define Vaddr VADDR
mobj_define Paddr PADDR
indxobj_define VIRTPC "VIRTPC"
indxobj_define PHYSPC "PHYSPC"
indxobj_define Process PID
indxobj_define Thread (PID*1000)+THRID
indxobj_define ThreadID THRID
indxobj_define Seconds (TSTAMP/1000000000)
indxobj_define Minutes (TSTAMP/60000000000)
mobj_define US4p_L1DataCacheLine (VADDR&0x3fe0)>>5
mobj_define US4p_L2CacheLine (PADDR&0x7ffc0)>>6
mobj_define US4p_L3CacheLine (PADDR&0x7ffc0)>>6
mobj_define VA_L2 VADDR>>6
mobj_define VA_L1 VADDR>>5
mobj_define PA_L2 PADDR>>6
mobj_define PA_L1 PADDR>>5
mobj_define US4p_T512_8k (VADDR&0x1fe000)>>13
mobj_define US4p_T512_64k (VADDR&0xff0000)>>16
mobj_define US4p_T512_512k (VADDR&0x7f80000)>>19
mobj_define US4p_T512_4M (VADDR&0x3fc00000)>>22
mobj_define US4p_T512_32M (VADDR&0x1fe000000)>>25
mobj_define US4p_T512_256M (VADDR&0xff0000000)>>28
mobj_define Vpage_32M VADDR>>25
mobj_define Vpage_256M VADDR>>28
mobj_define Ppage_32M PADDR>>25
mobj_define Ppage_256M PADDR>>28
```

Ultra SPARCIV+ の場合

お勧め Solaris 洋書

- ✓ Solaris Internals
 - ✓ kernel internal (Solaris10/OpenSolaris に対応)
- ✓ Solaris Performance and Tools
 - ✓ DTrace, MDB etc
- ✓ Solaris Application Programming
 - ✓ CMT, Compiler, Performance Analyzer etc

