Living on the Edge : Rapid Probe Toggling on x86

Buddhika Chamith, Bo Joel Svensson, Luke Dalessandro
Ryan R. Newton

How to modify a memory word atomically?

A data word

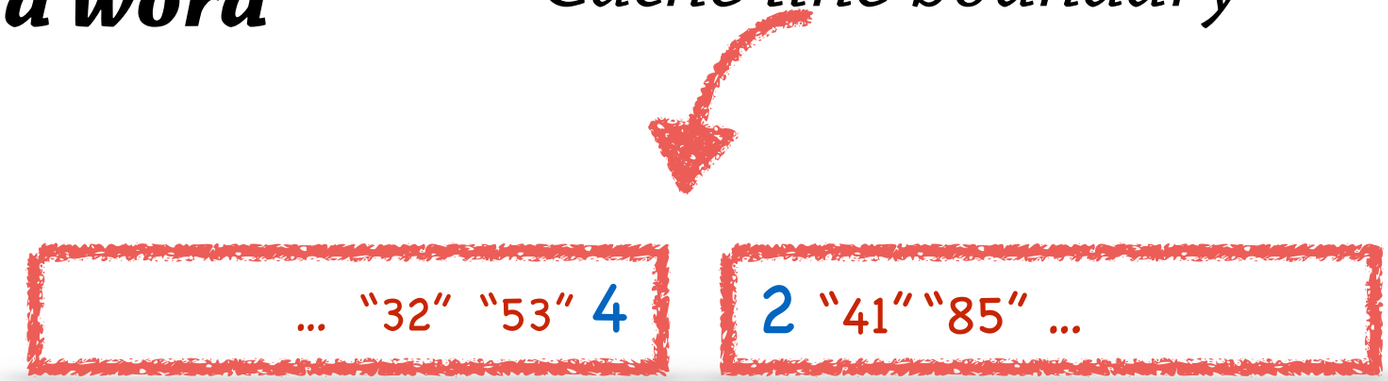
A data word

Compare and Swap (CAS)?

Asserts ***Lock*** signal on memory bus

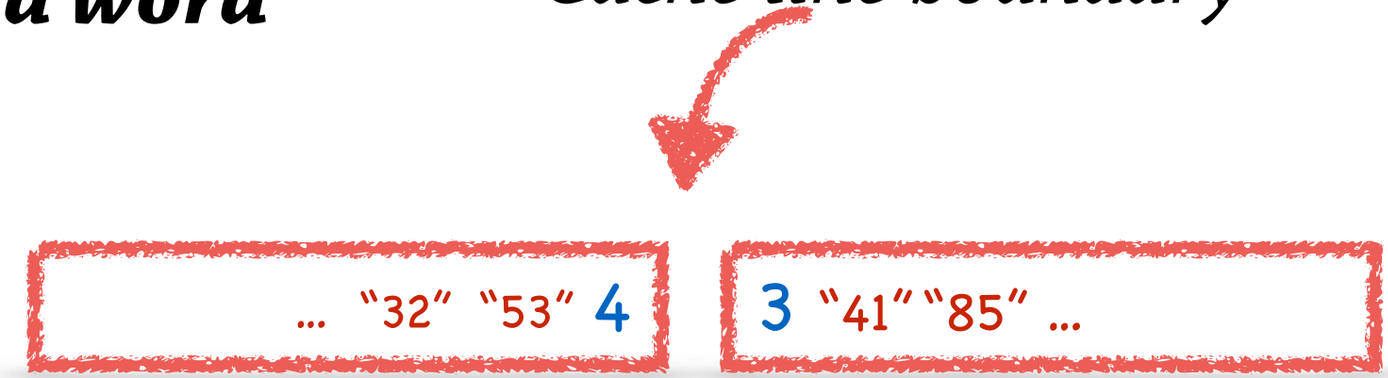
A data word

Cache line boundary



A data word

Cache line boundary

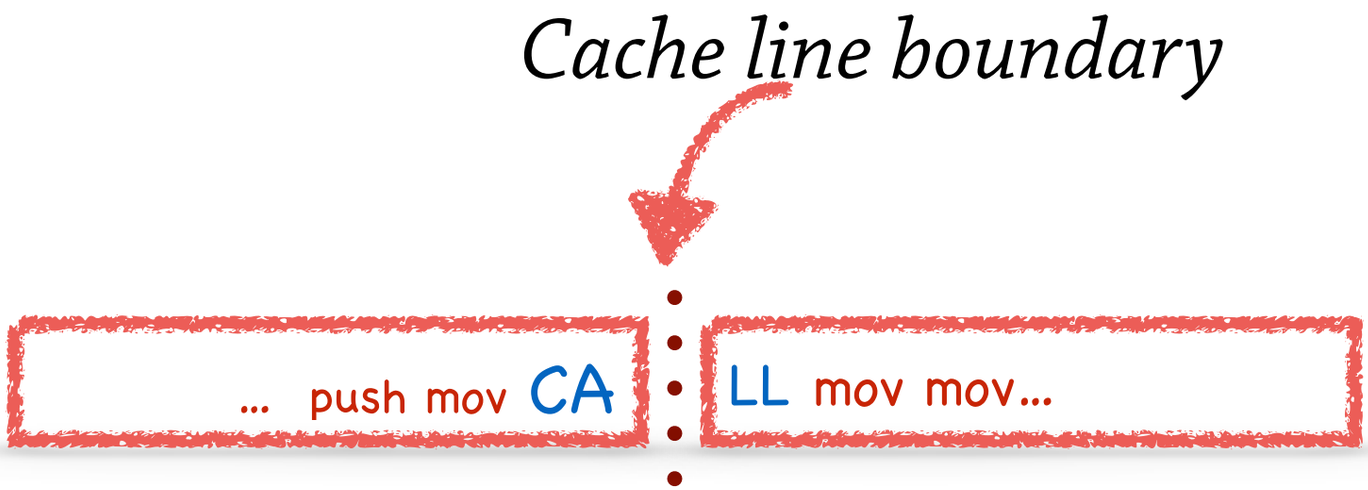


How to modify an instruction word atomically?

An instruction word

An instruction word

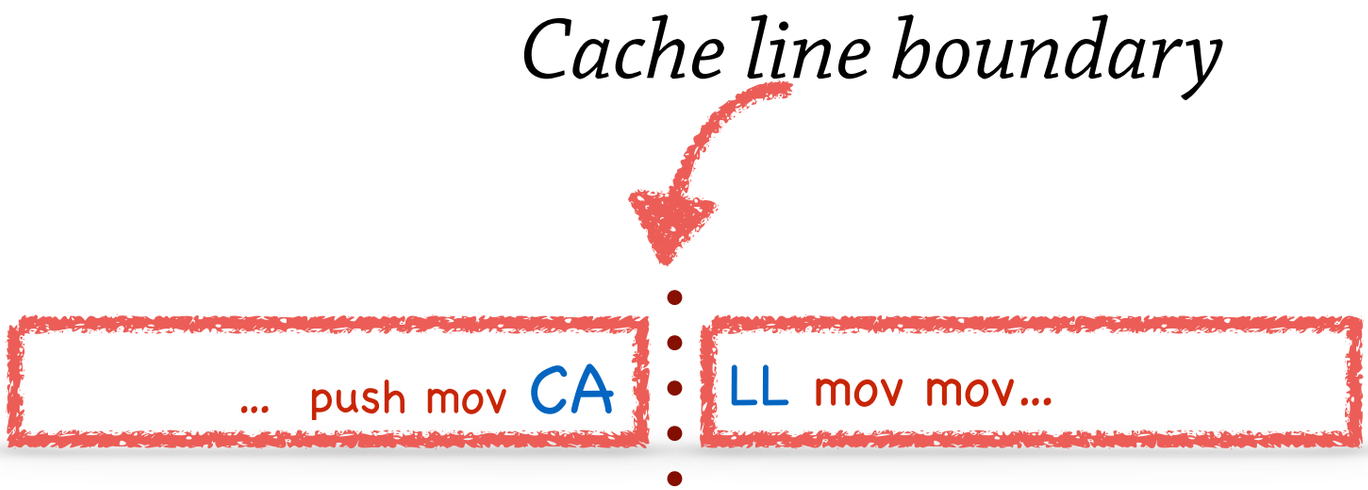
Woes of CAS



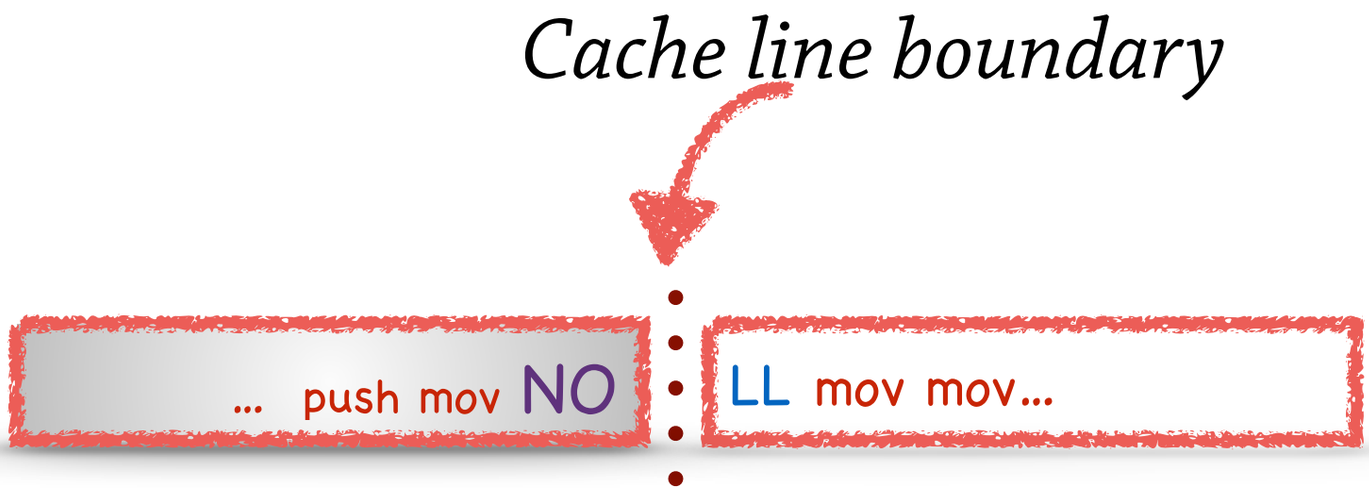
Woes of CAS

Locked operations are not atomic with respect to ***instruction fetch*** when they operate on multiple cache-lines.

Woes of CAS



Woes of CAS



INCORRECT

Cross Modification

Definition : The act of one processor writing data into the currently executing code segment of a second processor.

So how can we do cross modification safely?

We wanted to build a lightweight profiler..

Lightweight Profiler

- Instrumentation based

foo: ***<call instrument_fn>***
push
mov
…
<call instrument_fn>
ret

Lightweight Profiler

- Periodic Probing

foo: ***<call instrument_fn>***
push
mov
…
<call instrument_fn>
ret

Lightweight Profiler

- Periodic Probing

foo: ***noop***
push
mov
…
noop
ret

Requirements

Cheap

```
fn1:
        call
        ....
        call
        ret

fn2:    call
        ...
        call
        ret

        ...
```

```
        ...

fn3:    call
        ...
        call
        ret

fn4:    call
        ...
        call
        ret

        ...
```

```
        ...

        ...

fn5:    call
        ...
        call
        ret

        ...

fnN:    call
        ...
        call
        ret
```

Requirements

Cheap

fn1:
call
....
call
ret

fn2: ***noop***
...
noop
ret

...

...

fn3: ***call***
...
call
ret

fn4: ***noop***
...
noop
ret

...

...

fn5: ***call***
...
call
ret

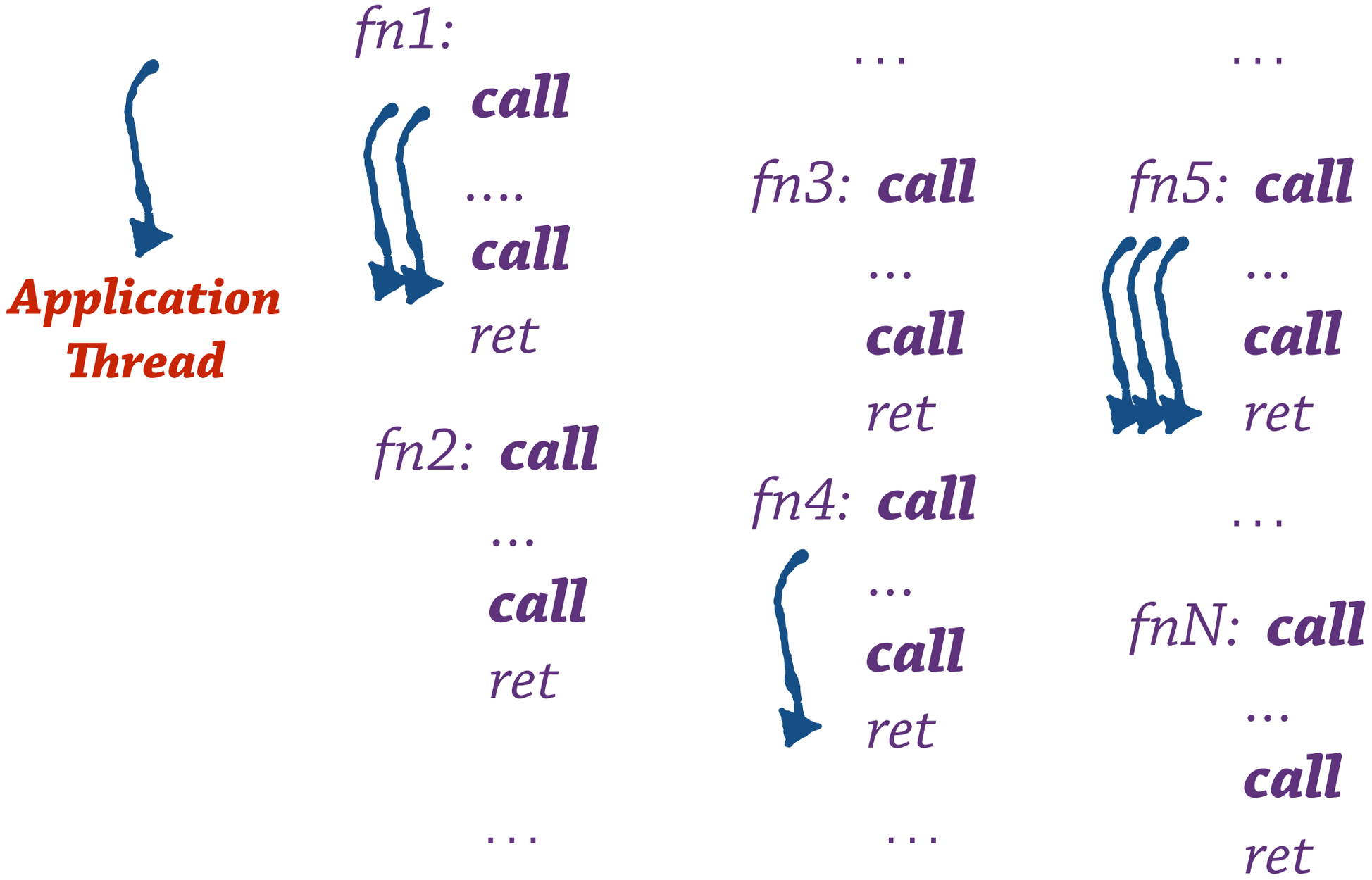
...

fnN: ***noop***
...
noop
ret

Requirements

Cheap

Thread Safe



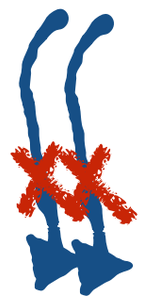
Requirements

Cheap

Thread Safe

Paused Thread

fn1:



call

....

call

ret

fn2: ***call***

...

call

ret

...

...

fn3: ***call***

...

call

ret

fn4: ***call***

...

call

ret

...

...

fn5: ***call***

...

call

ret

...

fnN: ***call***

...

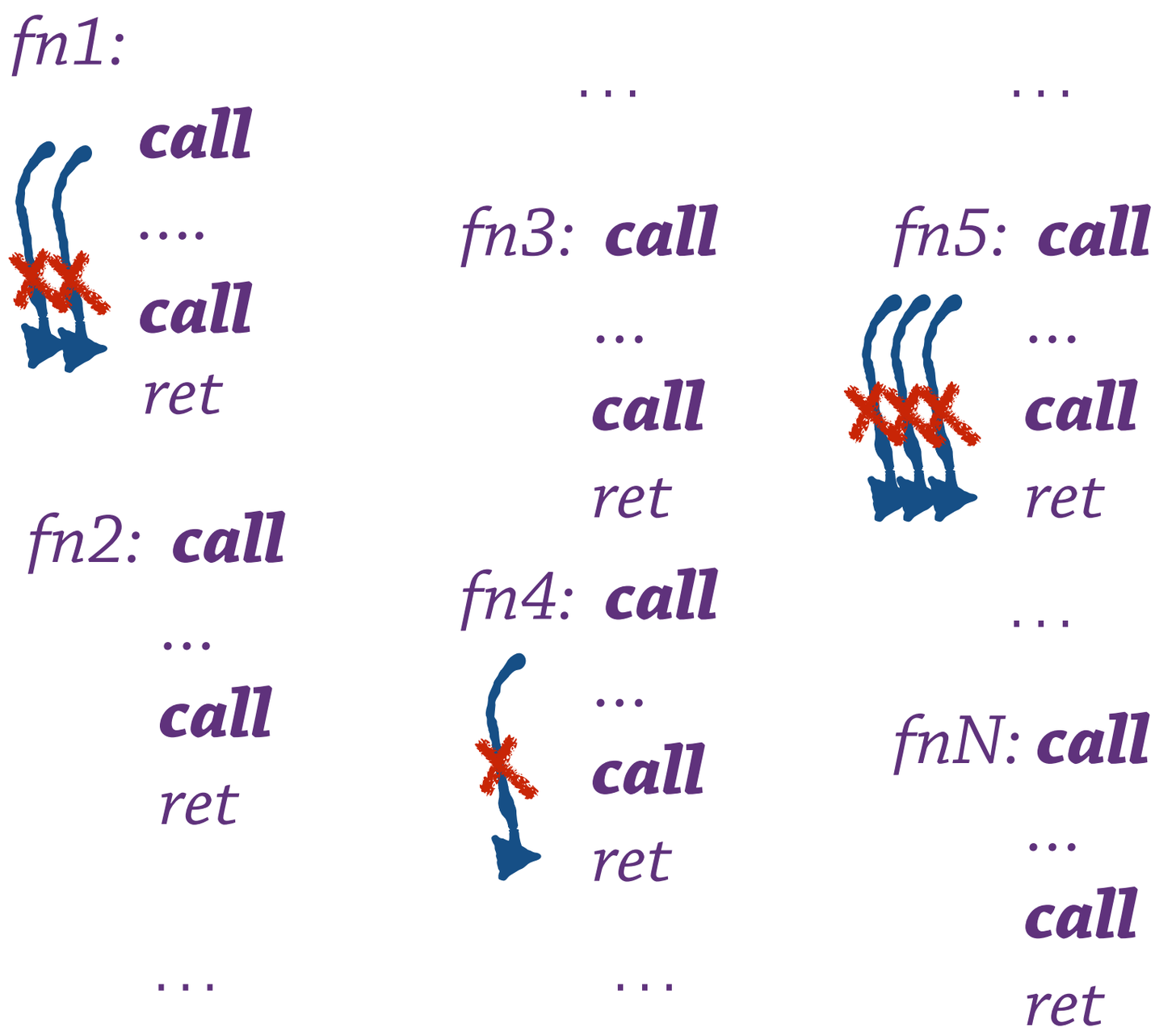
call

ret

Requirements

Cheap

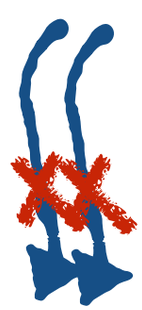
Thread Safe



Requirements

Cheap

Thread Safe



fn1:

noop

....

noop

ret

fn2: ***call***

...

call

ret

...

...

fn3: ***call***

...

call

ret

fn4: ***call***

...

call

ret

...

...

fn5: ***call***

...

call

ret

...

fnN: ***call***

...

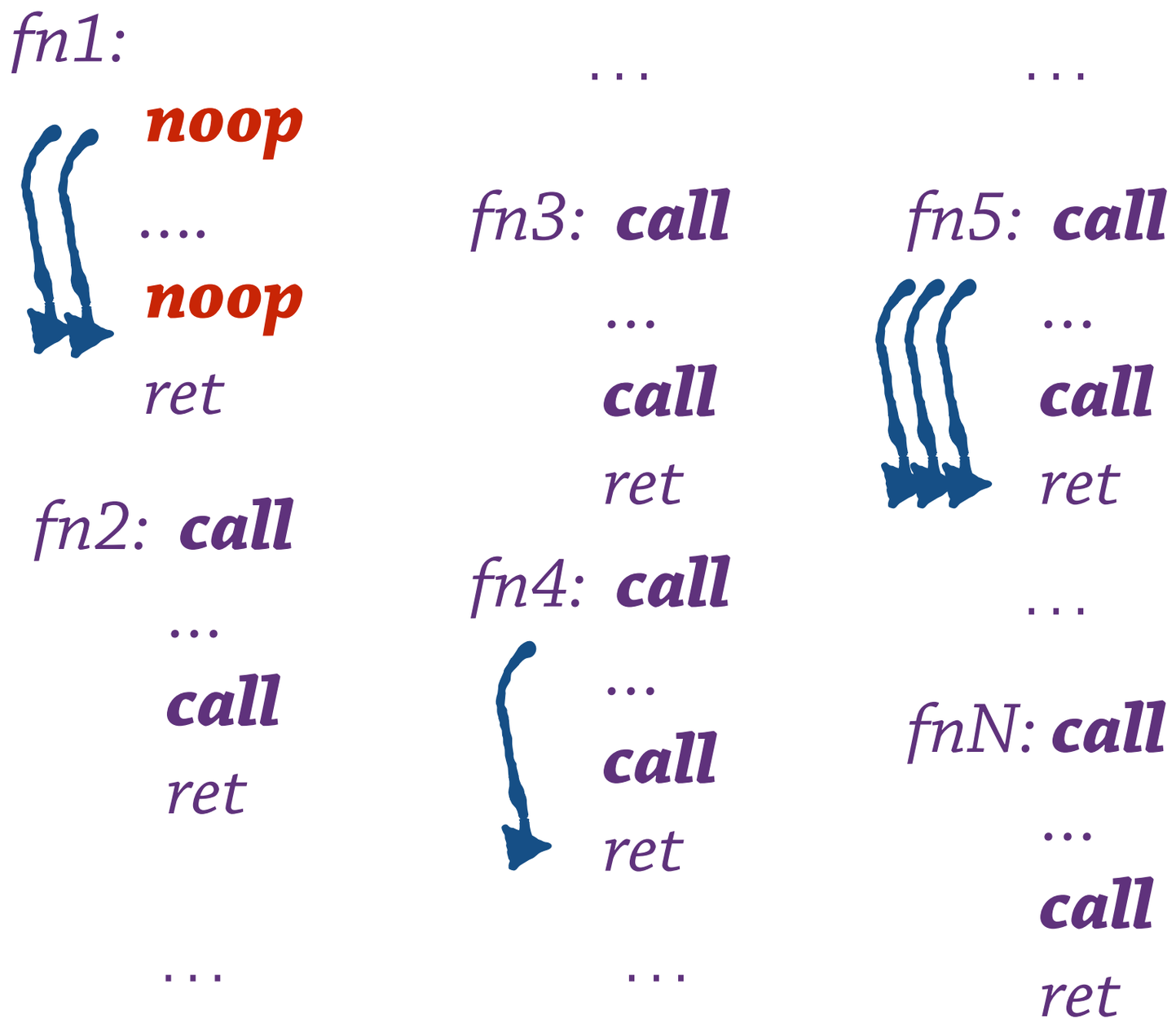
call

ret

Requirements

Cheap

Thread Safe



Requirements

Cheap

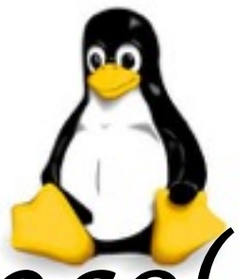
Thread Safe

Scalable

Image : http://goo.gl/yNtY60

Some Options :
In Place Instrumentation

Dyninst



ptrace(2)

× *Cheap*

✓ *Thread safe*

× *Scalable*

Dtrace

× *Cheap*

✓ *Thread safe*

✓ *Scalable*

Some Options :
In Place Instrumentation

Dyninst

ptrace(2)

× *Cheap*

✓ *Thread safe*

× *Scalable*

Dtrace

× *Cheap*

✓ *Thread safe*

✓ *Scalable*

Some Options (Contd): Out of Place Translation

Valgrind

× *Cheap*

✓ *Thread safe*

× *Scalable*

DynamoRIO

Assumption

There exists an upper bound for change propagation and visibility from other cores when doing cross modification.

T*max*

There is an upper bound T*max* where instruction cache/fetch & decode units are made eventually consistent after a write to code from one core/ processor.

Split Write Protocol (a.k.a. wordpatch)

Split Write Protocol (a.k.a. wordpatch)

Split Write Protocol (a.k.a. wordpatch)

A ***Tmax*** wait

Split Write Protocol (a.k.a. wordpatch)

A ***Tmax*** wait

Split Write Protocol (a.k.a. wordpatch)

Another ***Tmax*** wait.

Split Write Protocol (a.k.a. wordpatch)

Prerequisites

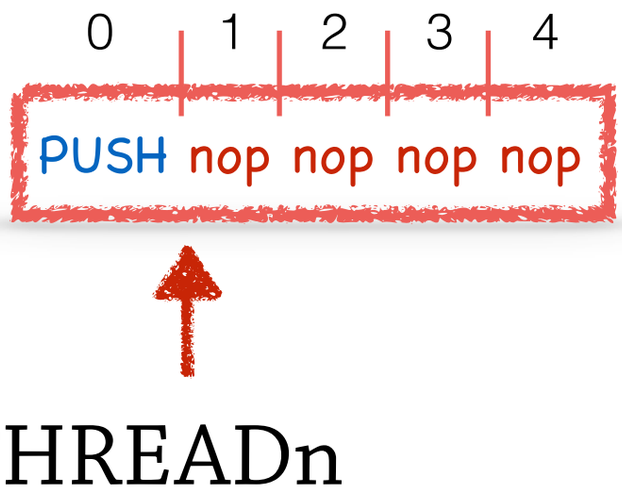
Writable code : Patch address must be writable.

Layout equivalence : Set of valid PC addresses remains invariant

No collisions : No addresses a1 and a2 may be concurrently modified if 0 <= |a1 - a2| <= *word size*

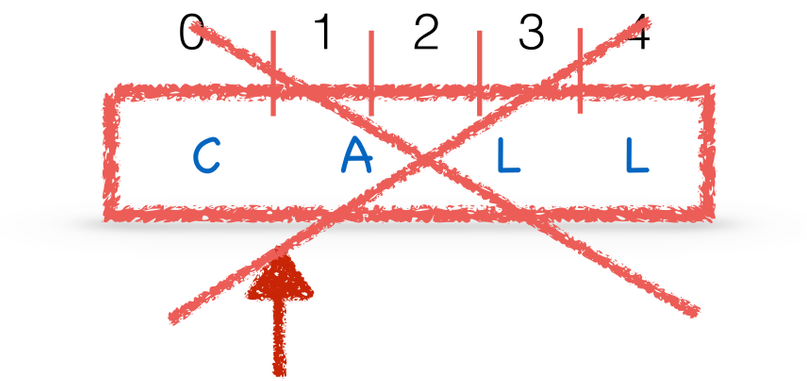
Layout Equivalence

Layout (in bytes) = *[1 |1| 1| 1| 1]*



Layout Equivalence

Layout (in bytes) = *[5]*



THREADn

FAIL

Validating T*max*

- ***A single thread toggling a call site***

- Stress test with multiple threads attempting to execute the call site.

- Many different configurations for
  - Straddle position
  - # of calling threads

- About 100 such test cases

Validating T*max*

- A single thread toggling a call site
- ***Stress test with multiple threads attempting to execute the call site.***
- Many different configurations for
  - Straddle position
  - # of calling threads
- About 100 such test cases

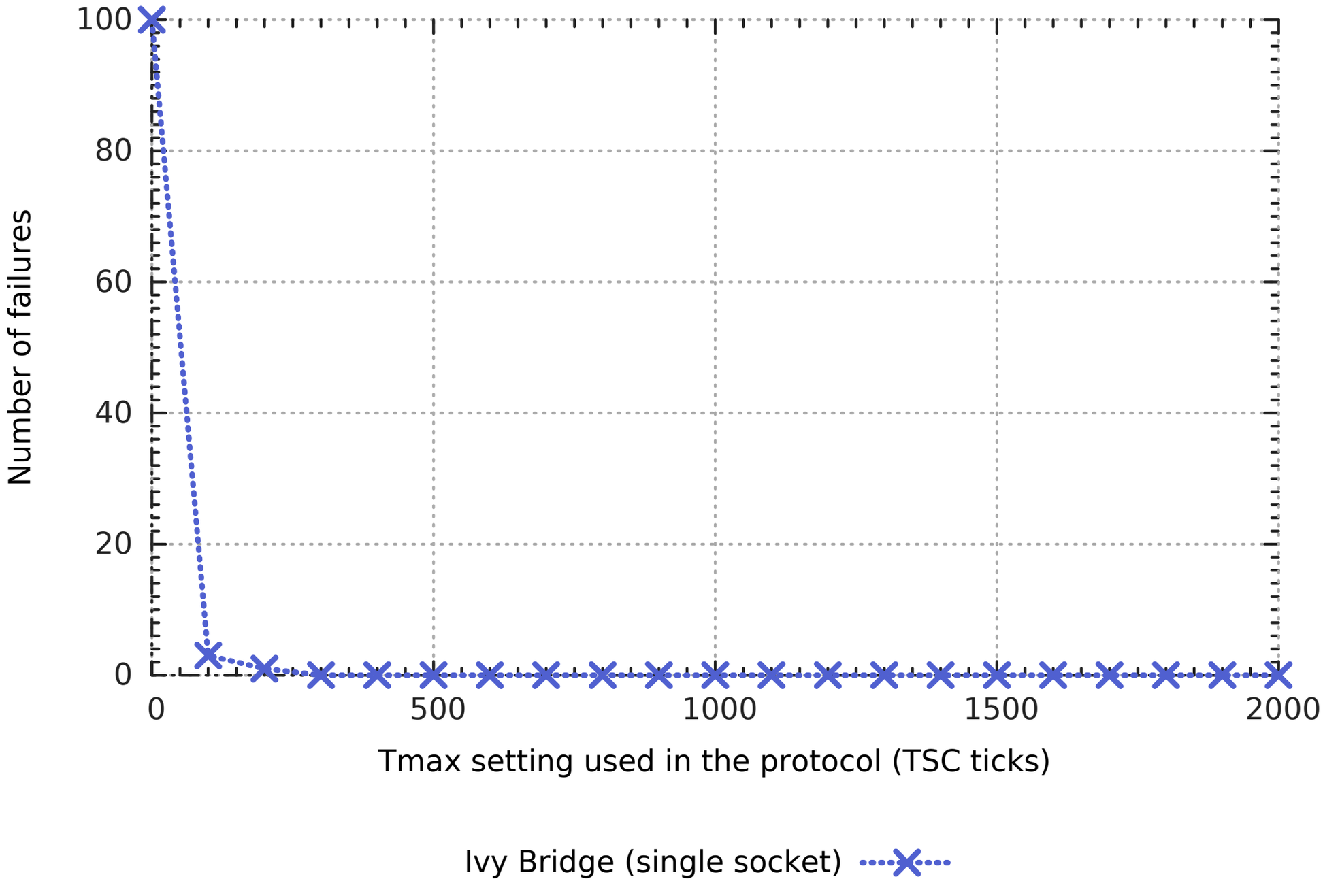
Validating T*max*

- A single thread toggling a call site
- Stress test with multiple threads attempting to execute the call site.
- ***Many different configurations for***
  - ***Straddle position***
  - ***# of calling threads***
- About 100 such test cases

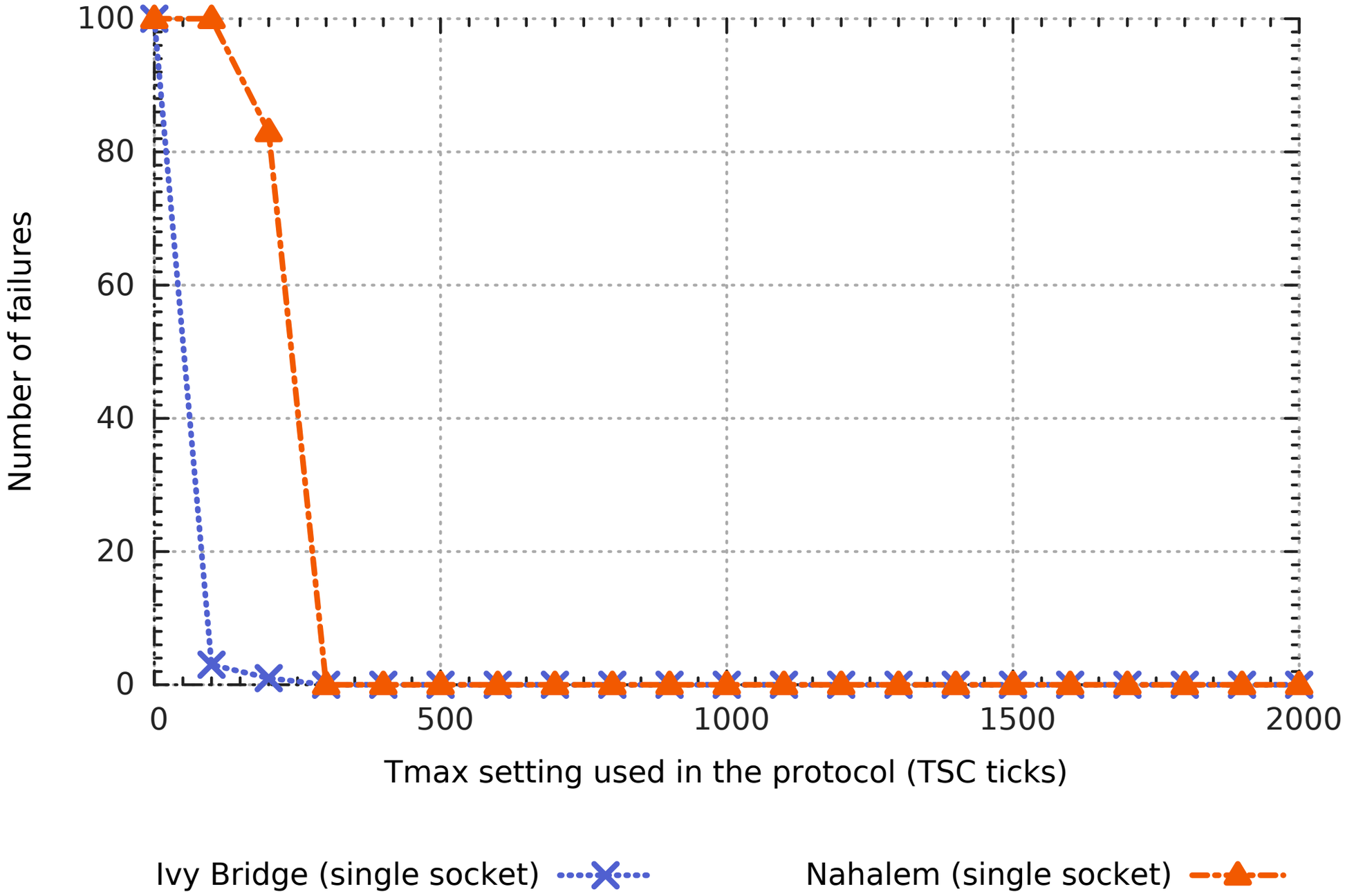
Validating T*max*

- A single thread toggling a call site
- Stress test with multiple threads attempting to execute the call site.
- Many different configurations for
  - Straddle position
  - # of calling threads
- ***About 100 such test cases***

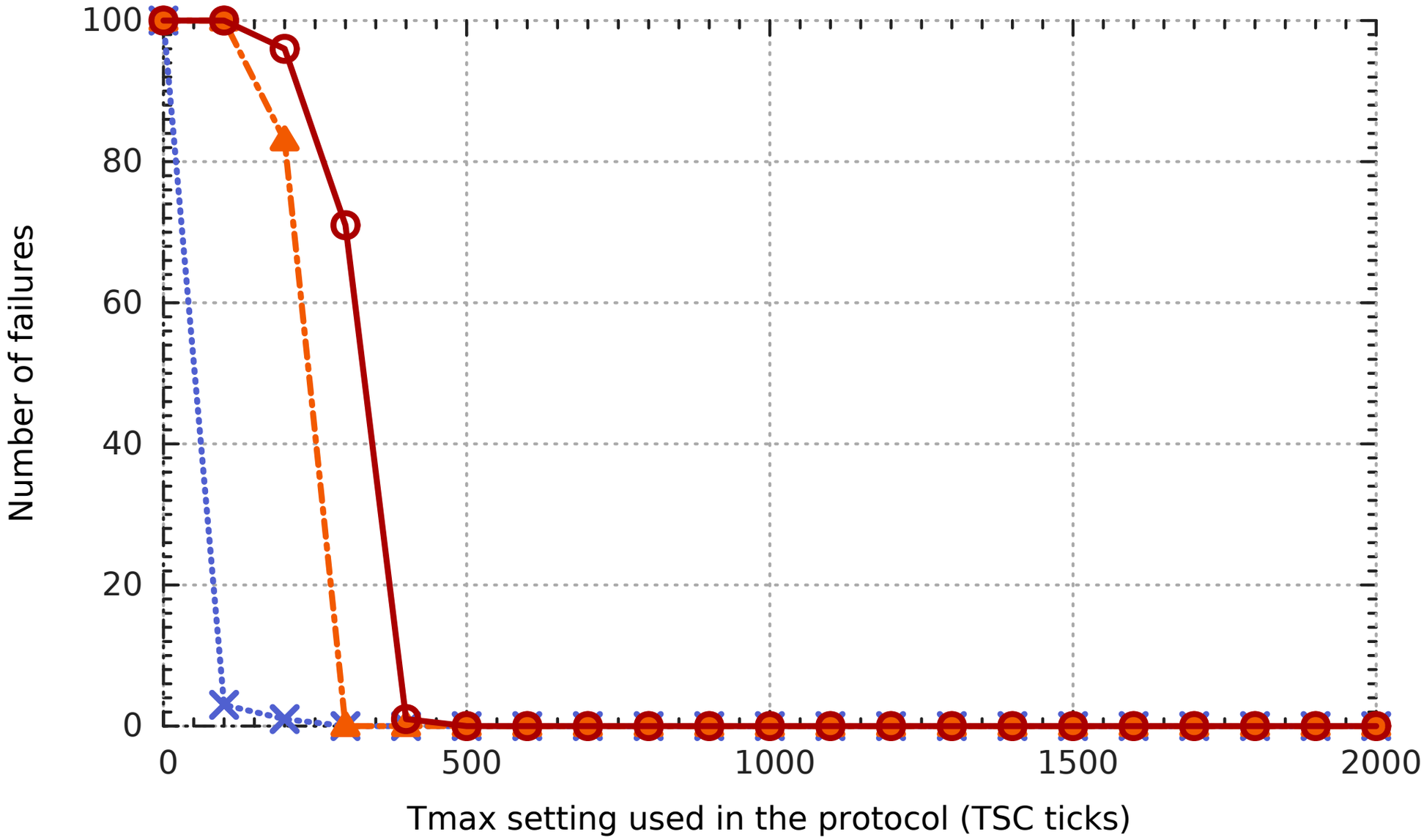
Validating T*max*



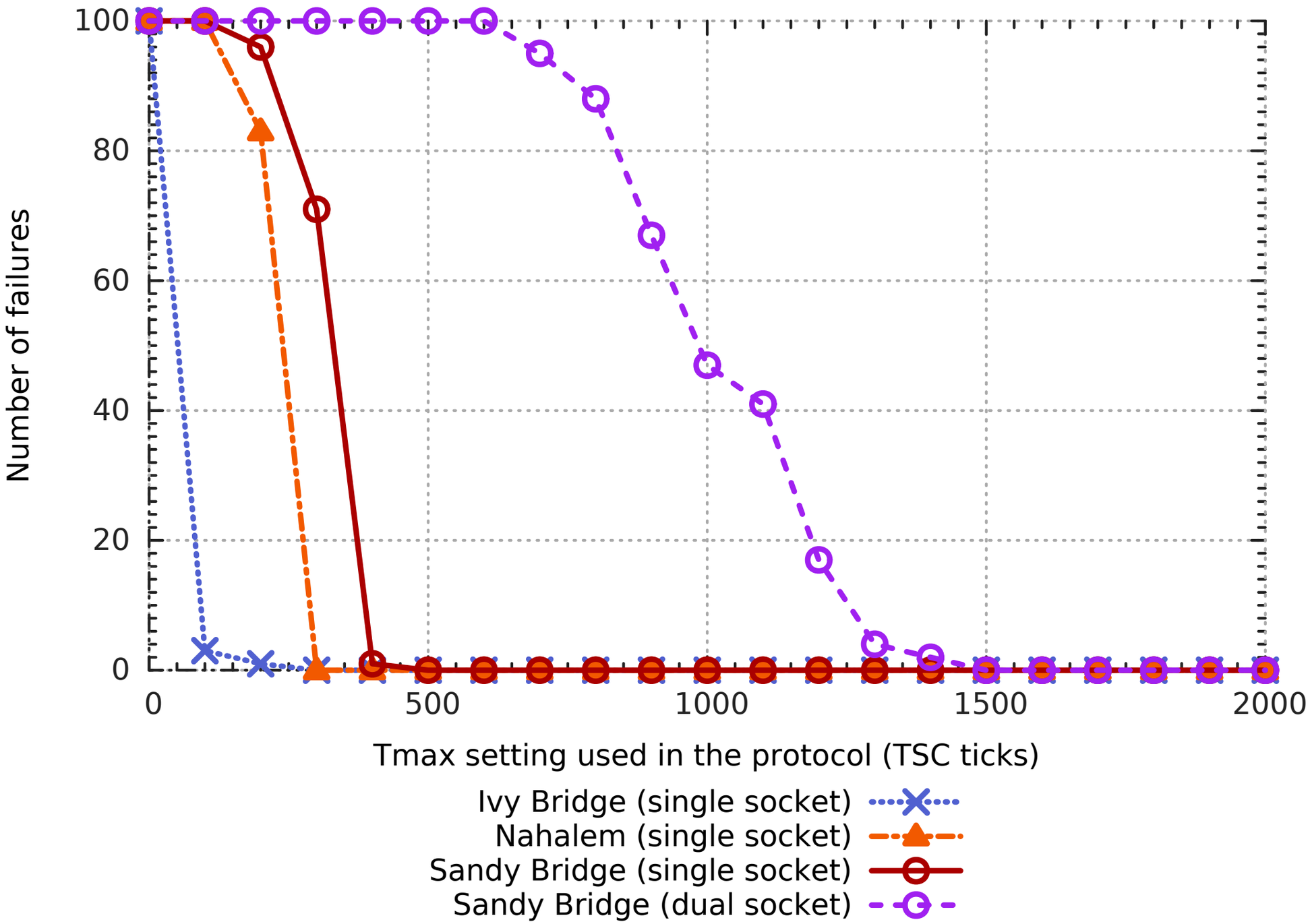
Validating T*max*



Validating T*max*



Validating T*max*



Safe ***wordpatch***

```
bool patch(address, value)
  if (not is_straddler(address))
    write(address, value);
    return true;
  else if (int3_lock(address))
    wait();
    write_back(address, value);
    wait();
    write_front(address, value);
    return true;
  else return false;
```

Safe *wordpatch*

```
//    Write an interrupt byte into the address,
void int3_lock(address)
  return write_front(address, INT3);
```

```
bool patch(address, value)
  if (not is_straddler(address))
    write(address, value);
    return true;
  else if (int3_lock(address))
    wait();
    write_back(address, value);
    wait();
    write_front(address, value);
    return true;
  else return false;
```

Safe *wordpatch*

```
//    Write an interrupt byte into the address,
void int3_lock(address)
  return write_front(address, INT3);

bool patch(address, value)
  if (not is_straddler(address))
    write(address, value);
    return true;
  else if (int3_lock(address))
    wait();
    write_back(address, value);
    wait();
    write_front(address, value);
    return true;
  else return false;
```

No global thread quiescence

Call Toggling (***callpatch***)

If only CALLs are patched

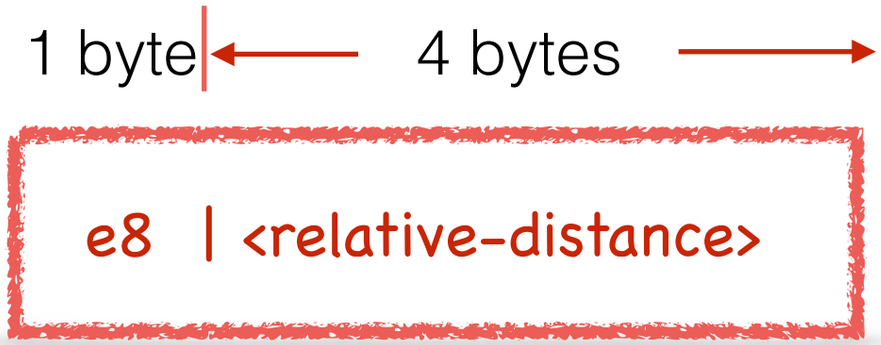
no Tmax/ Wait *necessary!*

Consider each straddling position ***separately***
and **patch only *one* half**

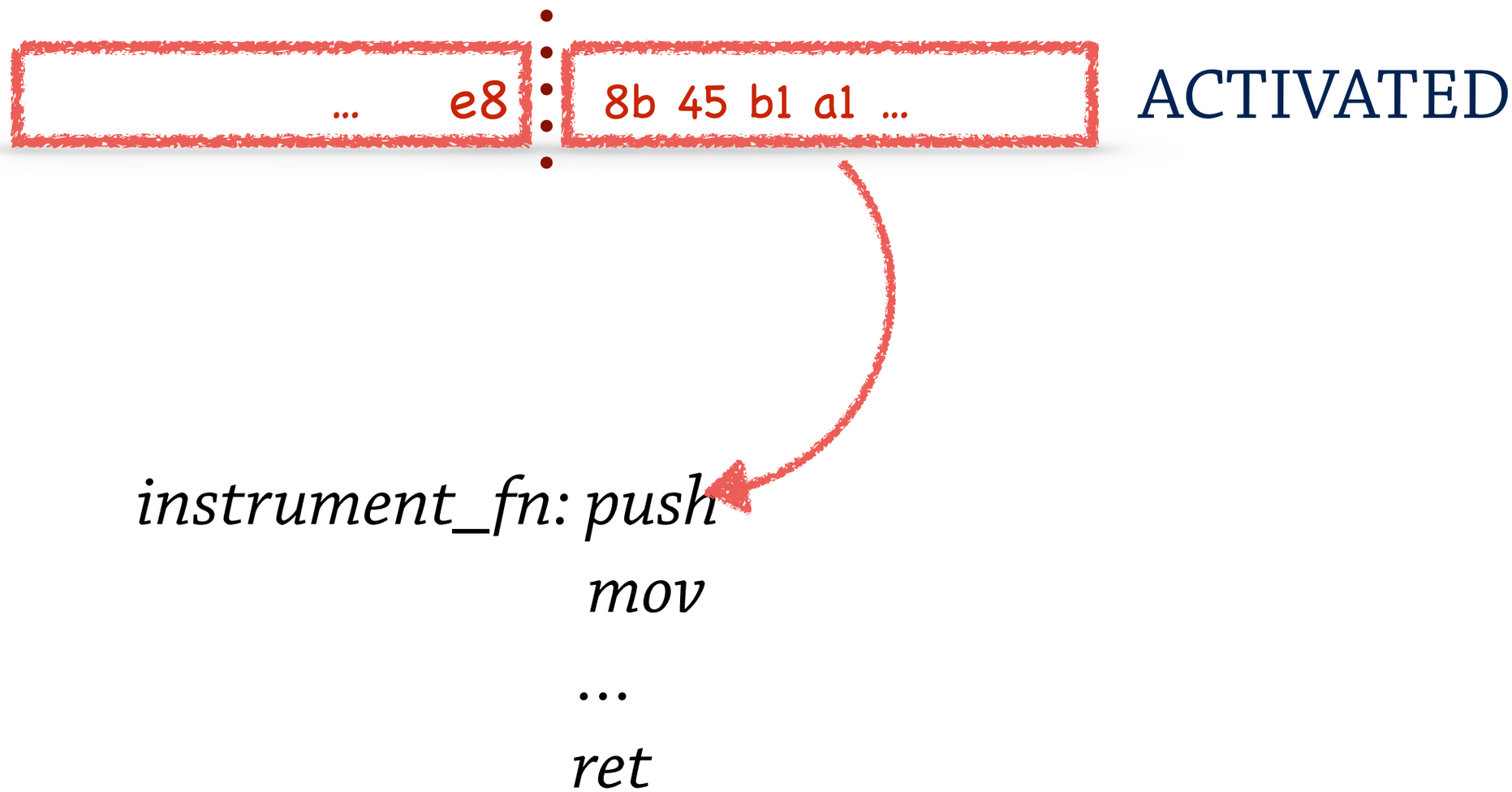
CALL Anatomy

Op-code | Operands

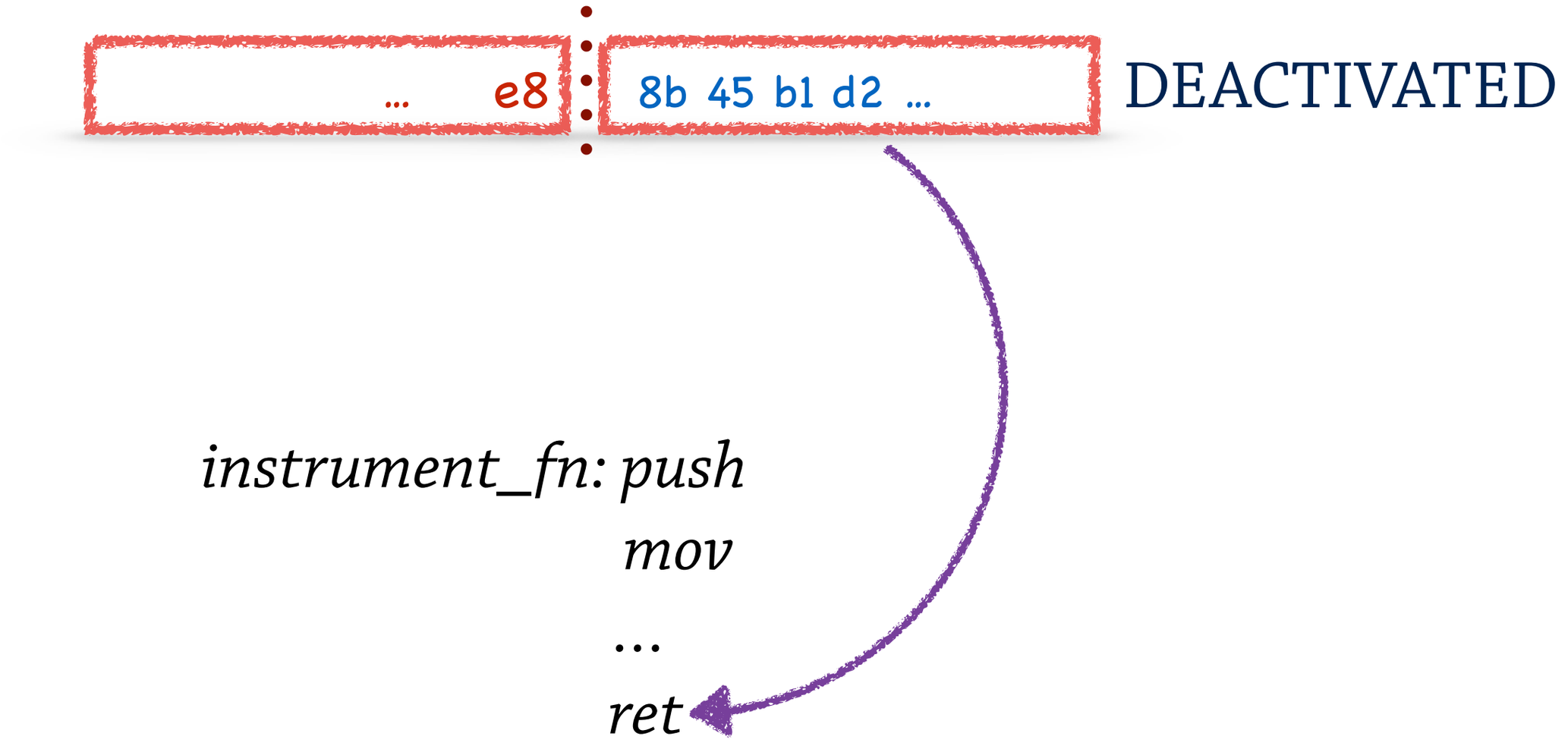
CALL Anatomy



[1|4] split

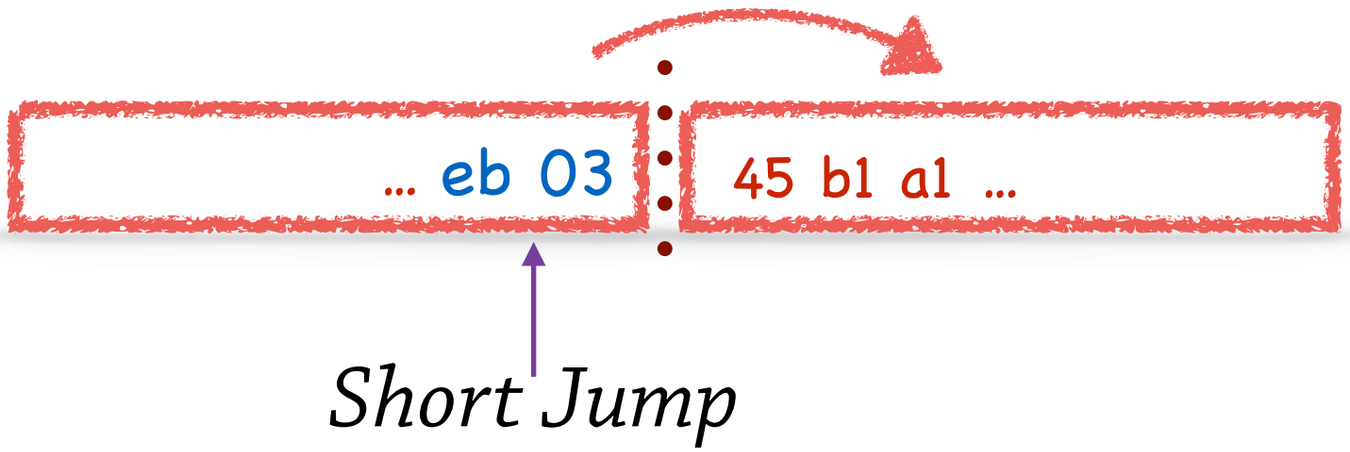


[1|4] split



[2|3] split

[2|3] split



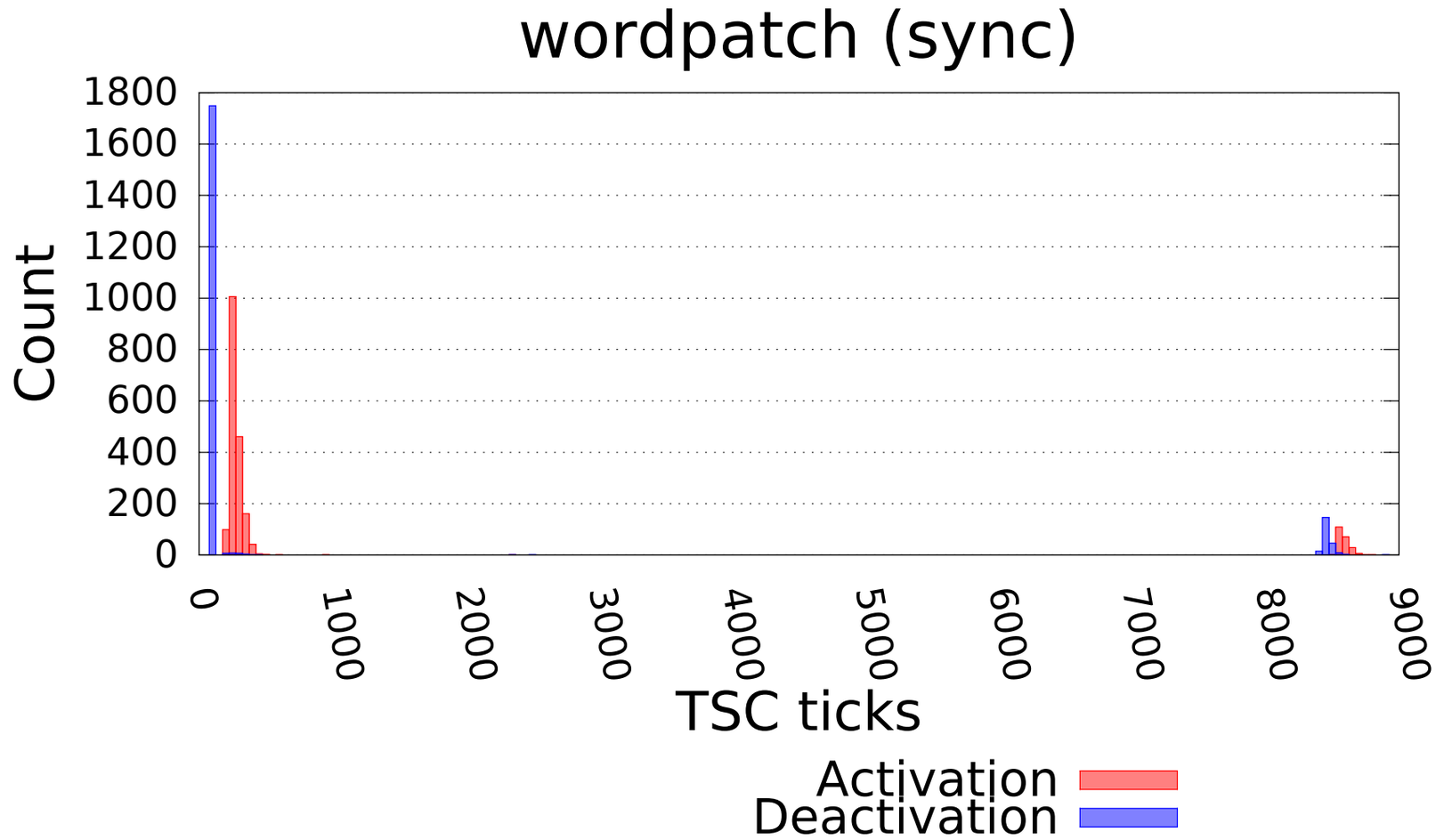
[3|2] & [4|1] splits

Reduces to [2|3] split

short jump scheme

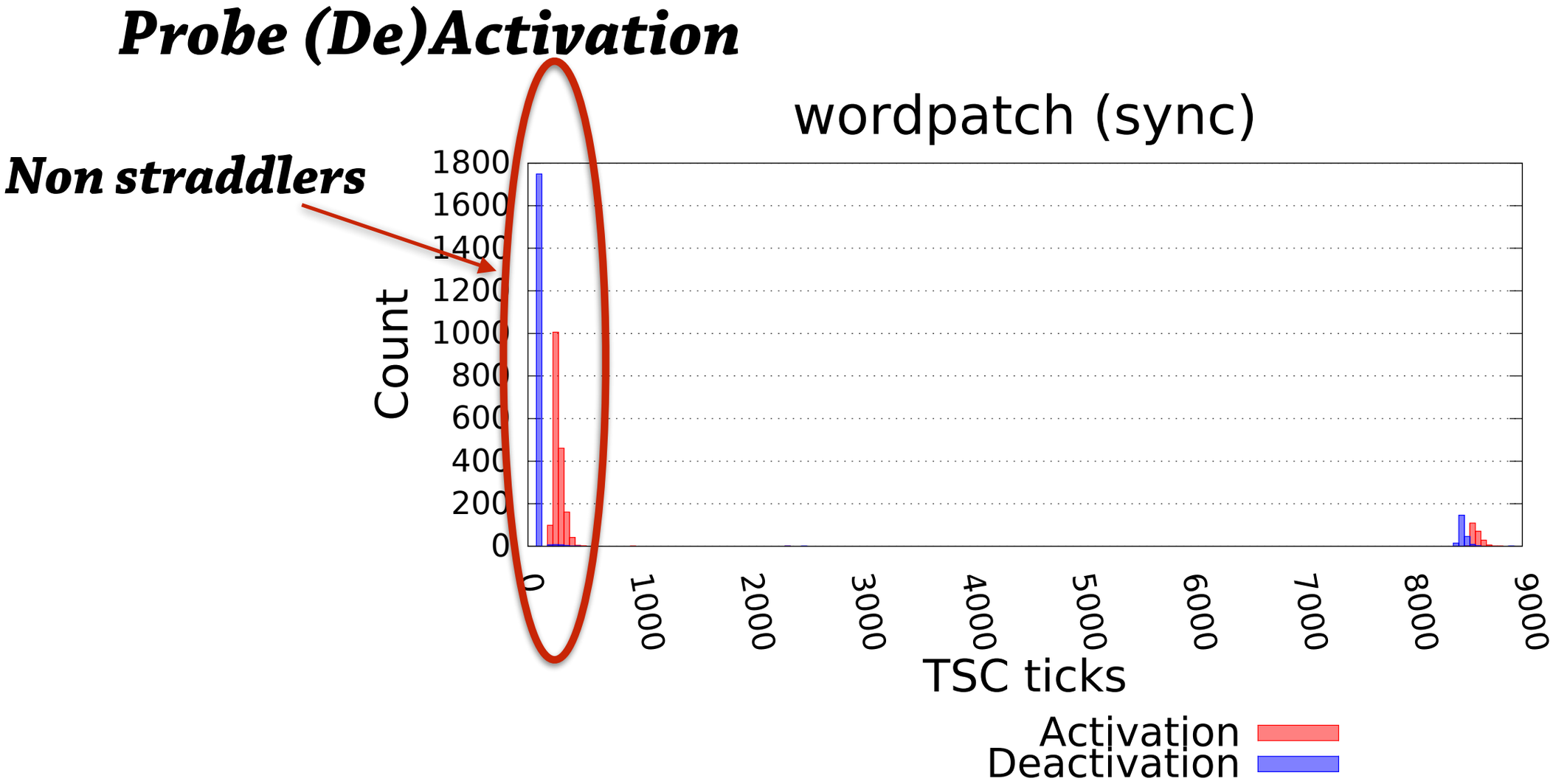
Probing Costs

Probe (De)Activation



On a synthetic application with 20000 probes

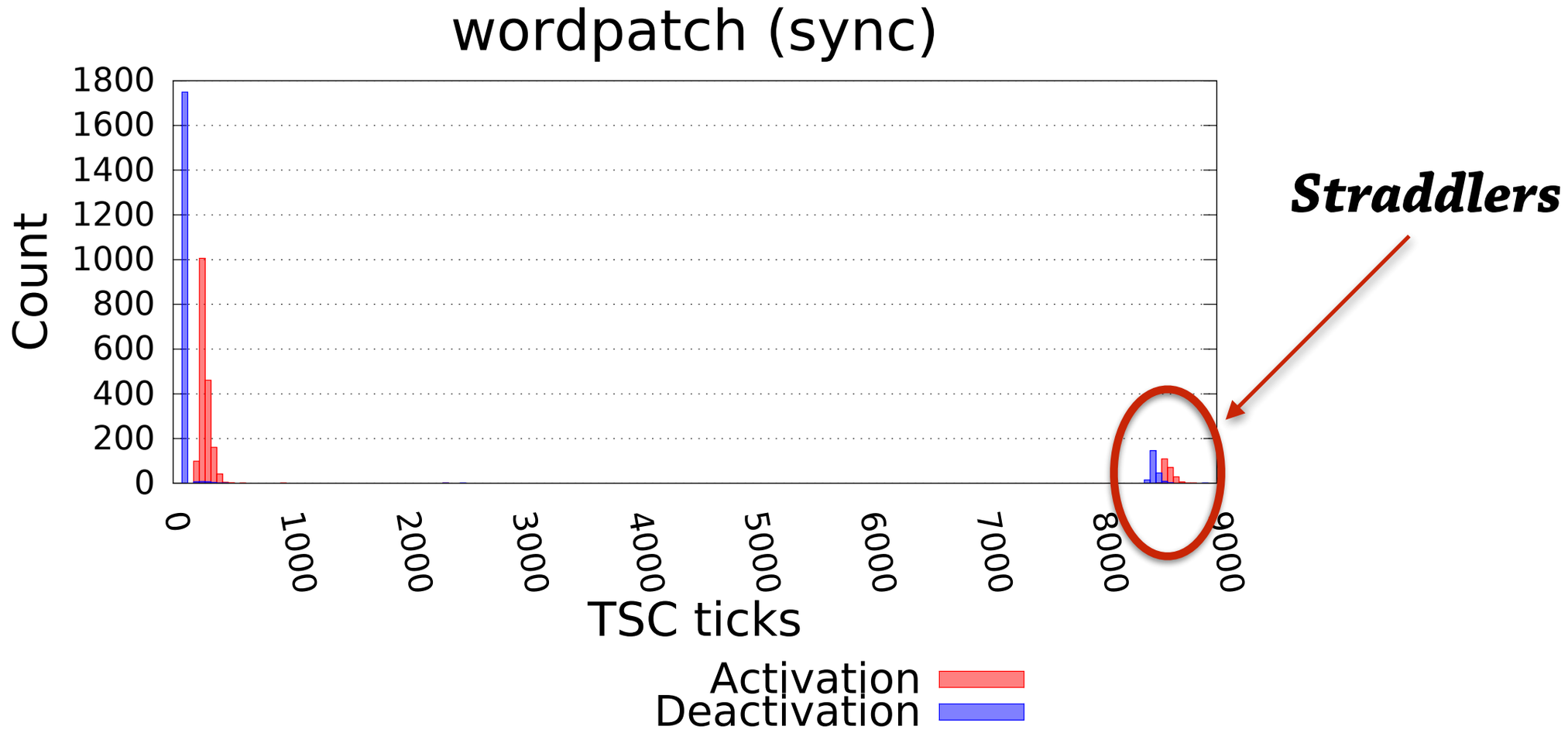
Probing Costs



On a synthetic application with 20000 probes

Probing Costs

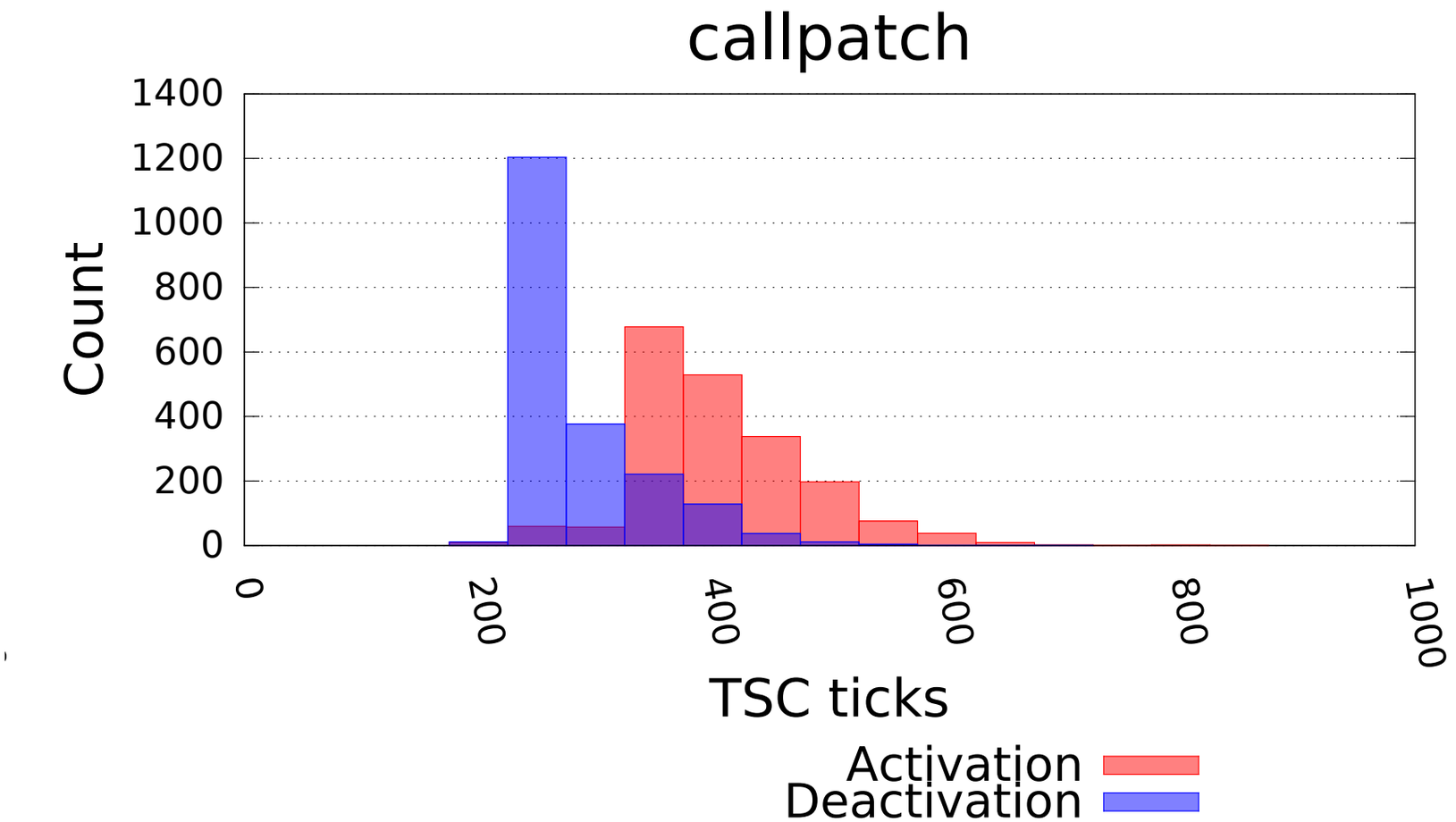
Probe (De)Activation



On a synthetic application with 20000 probes

Probing Costs

Probe (De)Activation



Comparison with Other Options

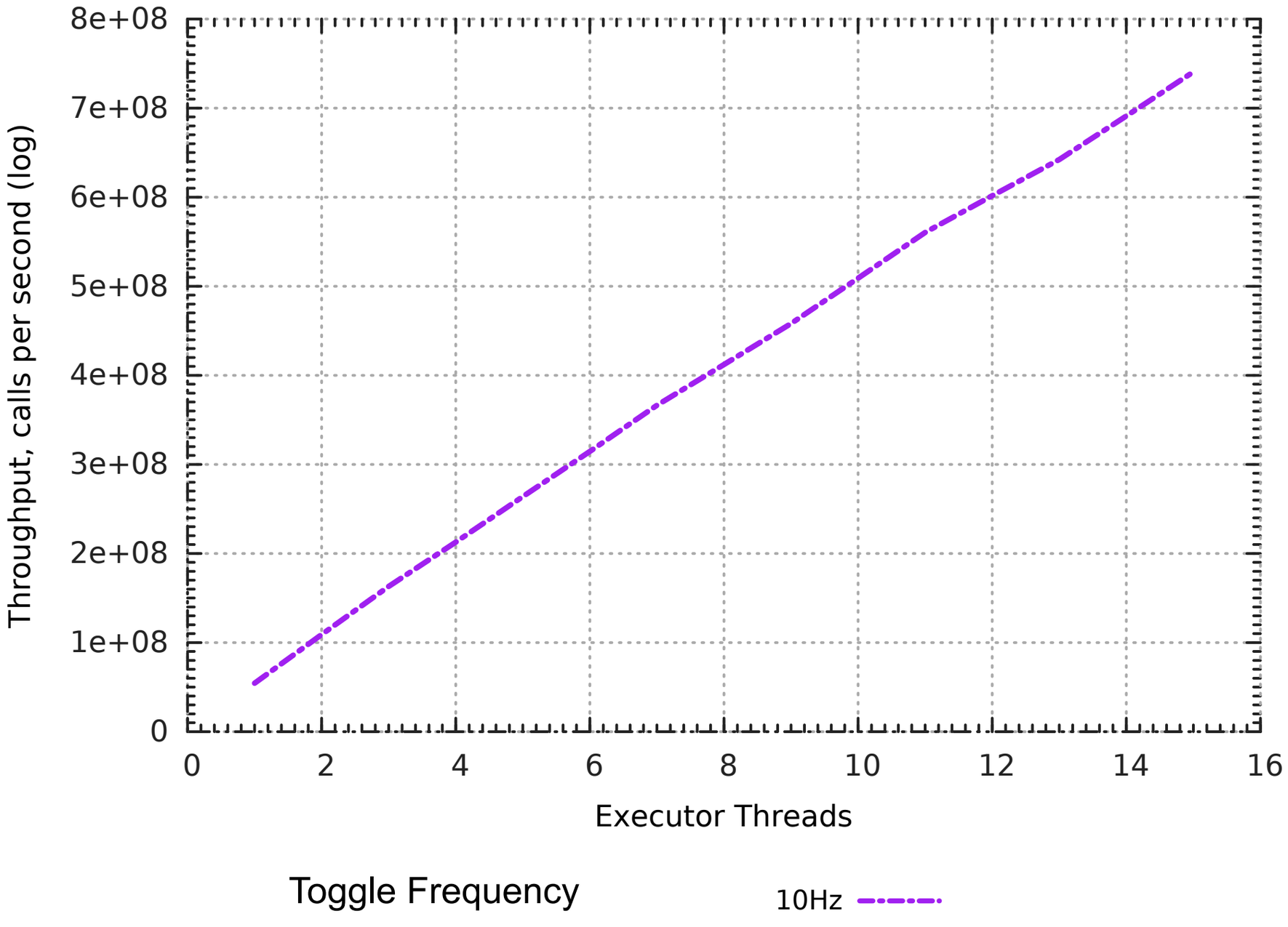
Method	Activation	Deactivation
wordpatch (non straddlers)	300	120
wordpatch(straddlers)	8850	8435
callpatch	432	315
JVM VolatileCallSite	995	995
Dyninst	1,929,244	995,447

Comparison with Other Options

Method	Activation	Deactivation
wordpatch (non straddlers)	300	120
wordpatch(straddlers)	8850	8435
callpatch	432	315
JVM VolatileCallSite	995	995
Dyninst	1,929,244	995,447

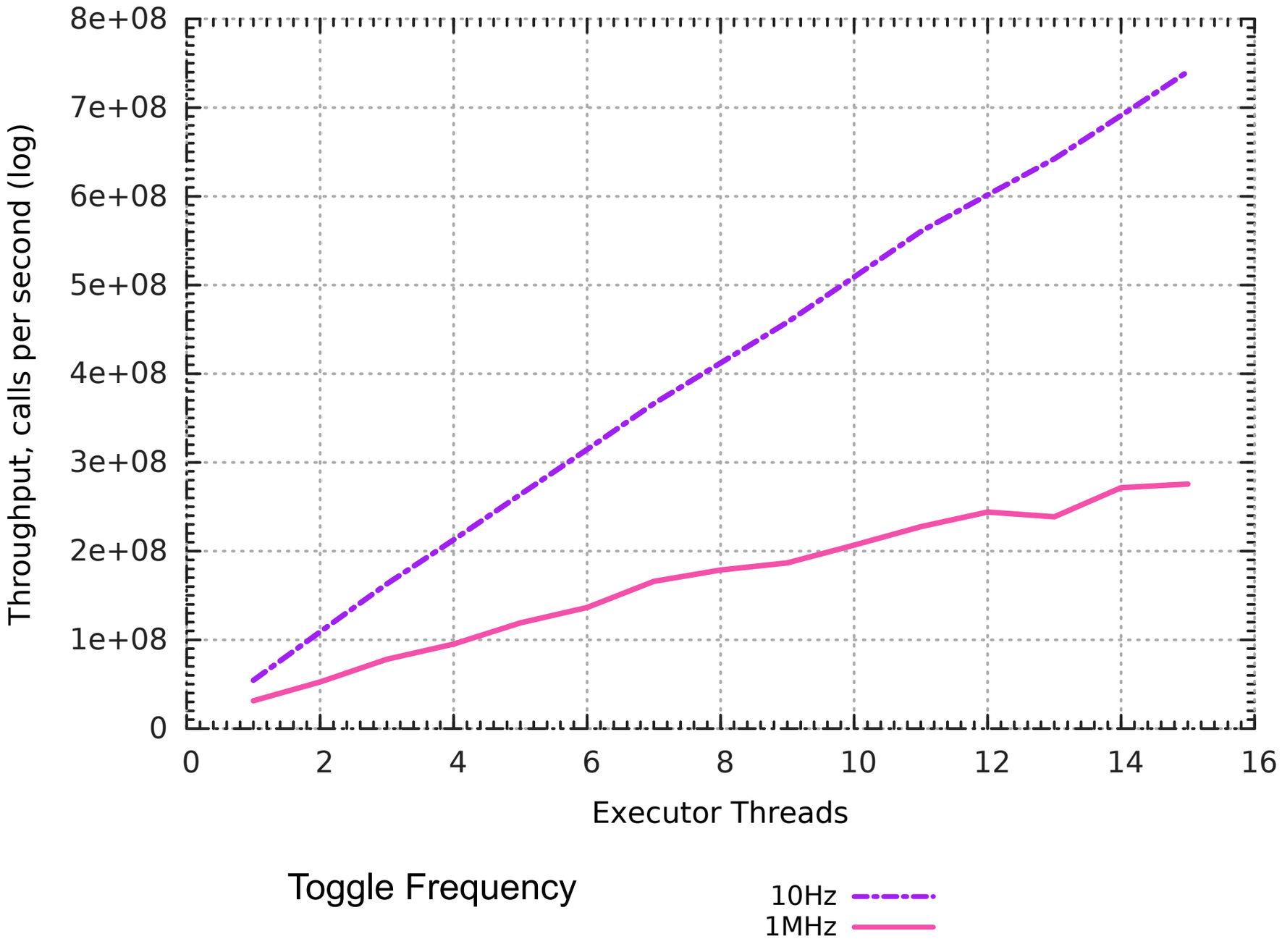
Scalability

Callpatch



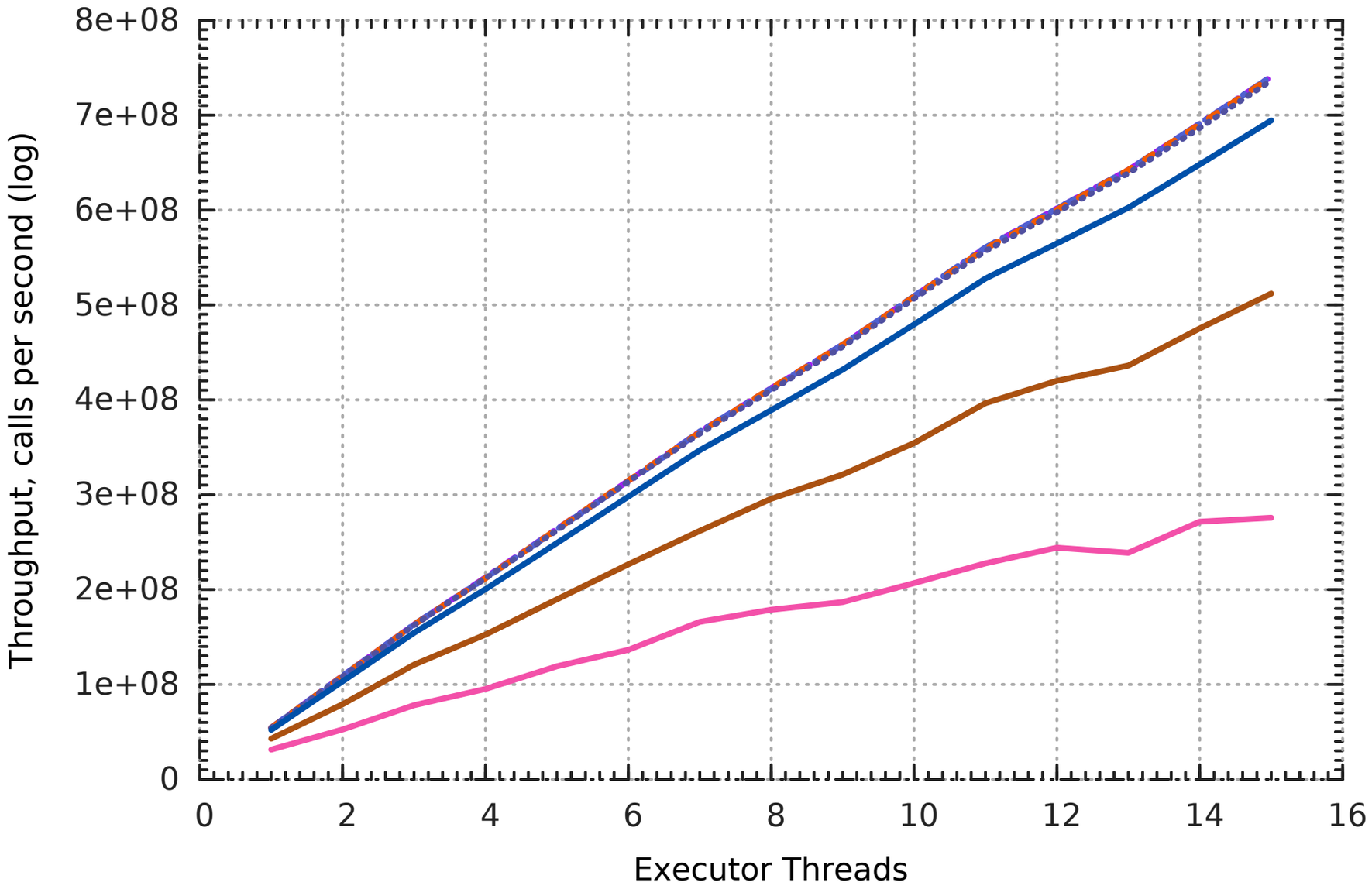
Scalability

Callpatch



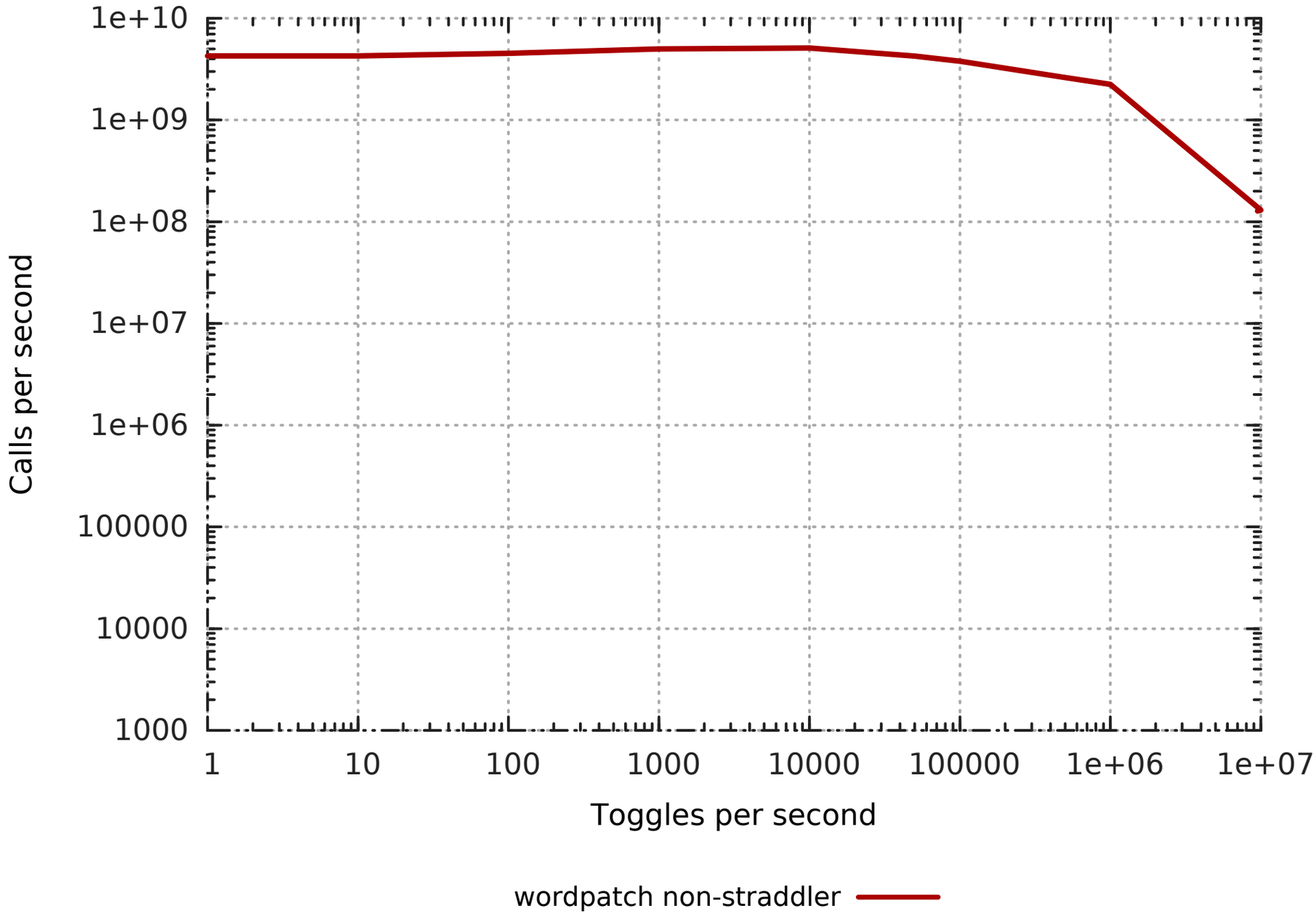
Scalability

Callpatch



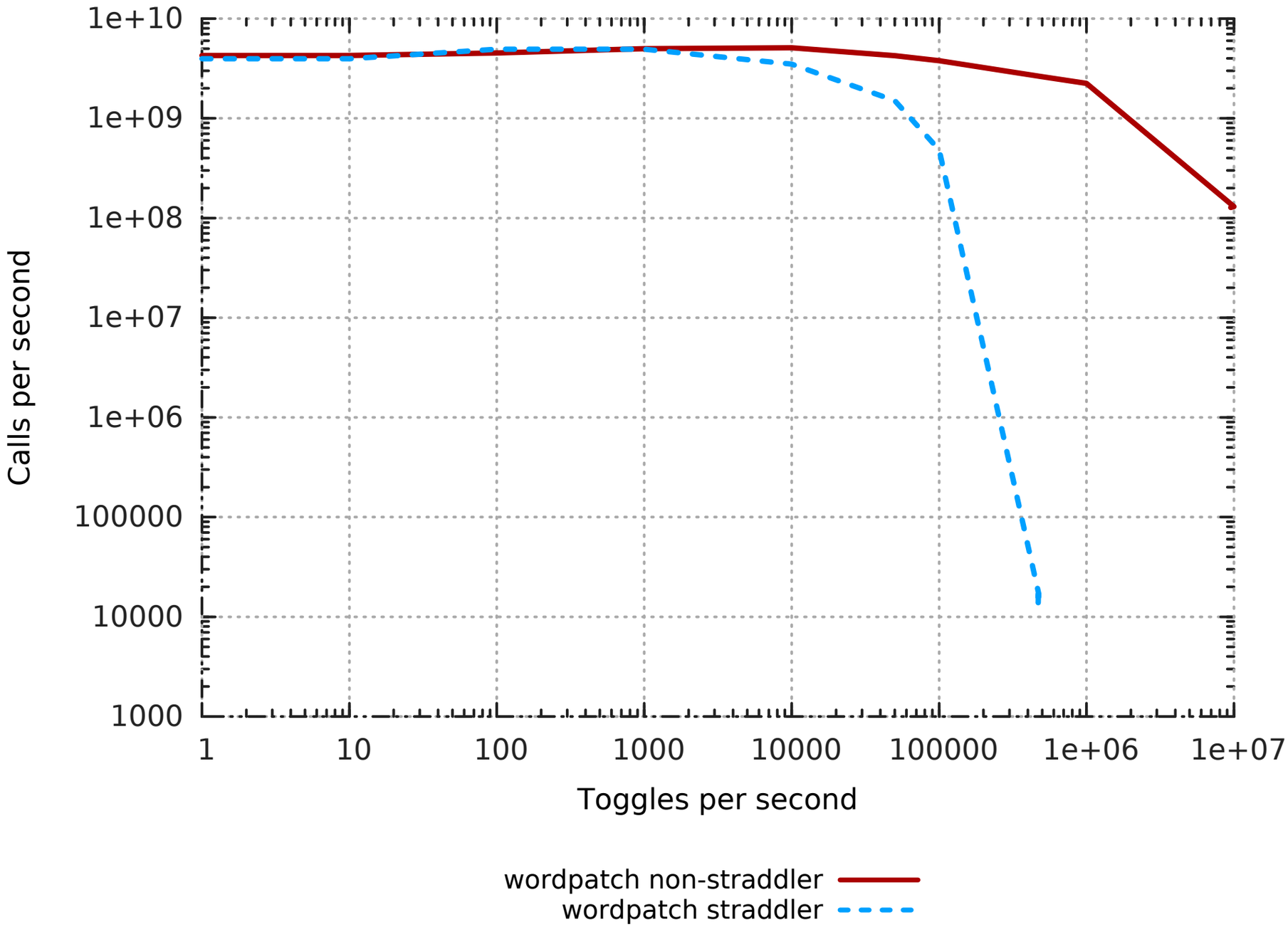
Comparison

Wordpatch



Comparison

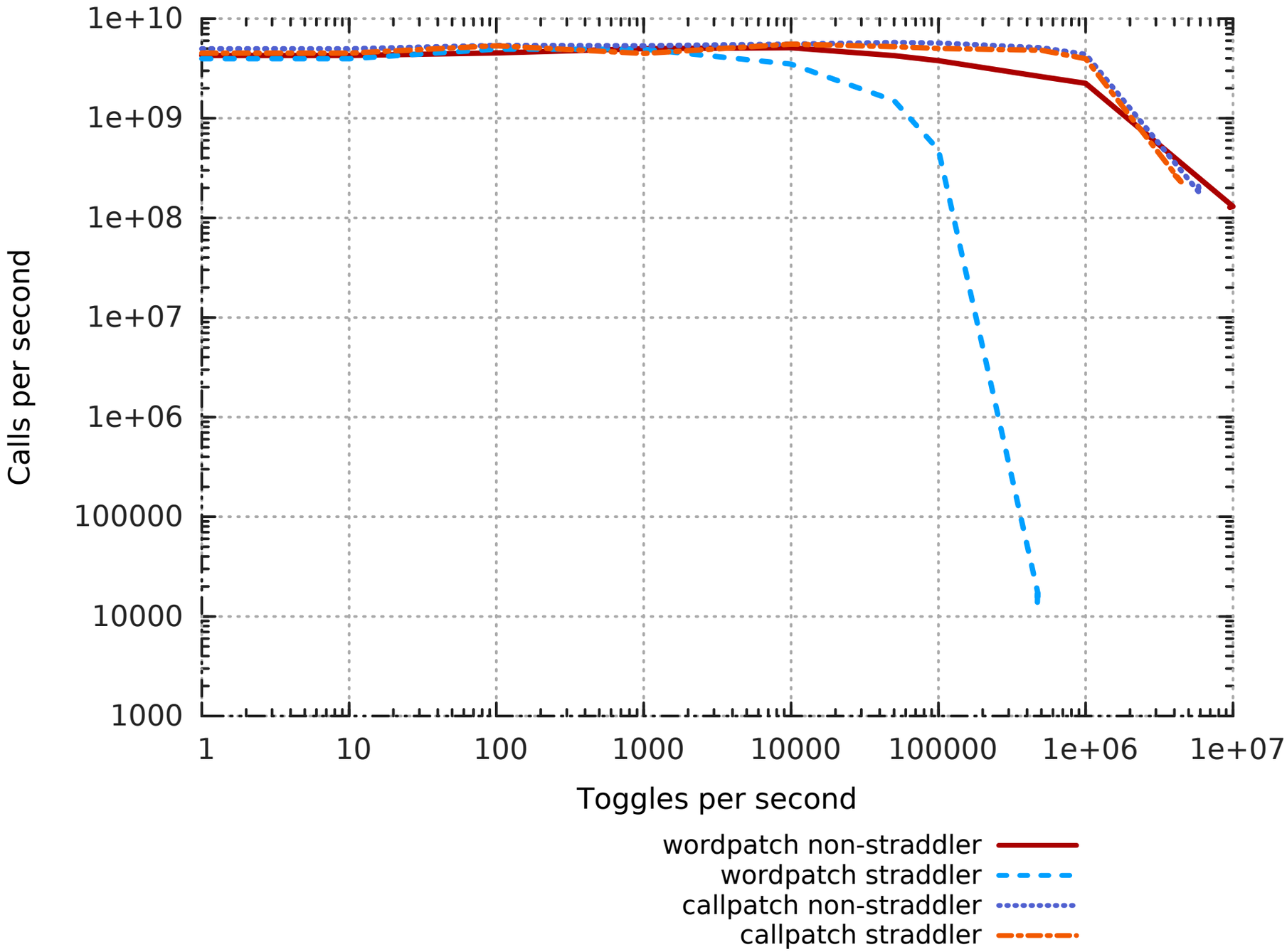
Wordpatch



Throughput is largely unaffected through ***100Khz*** *frequency.*

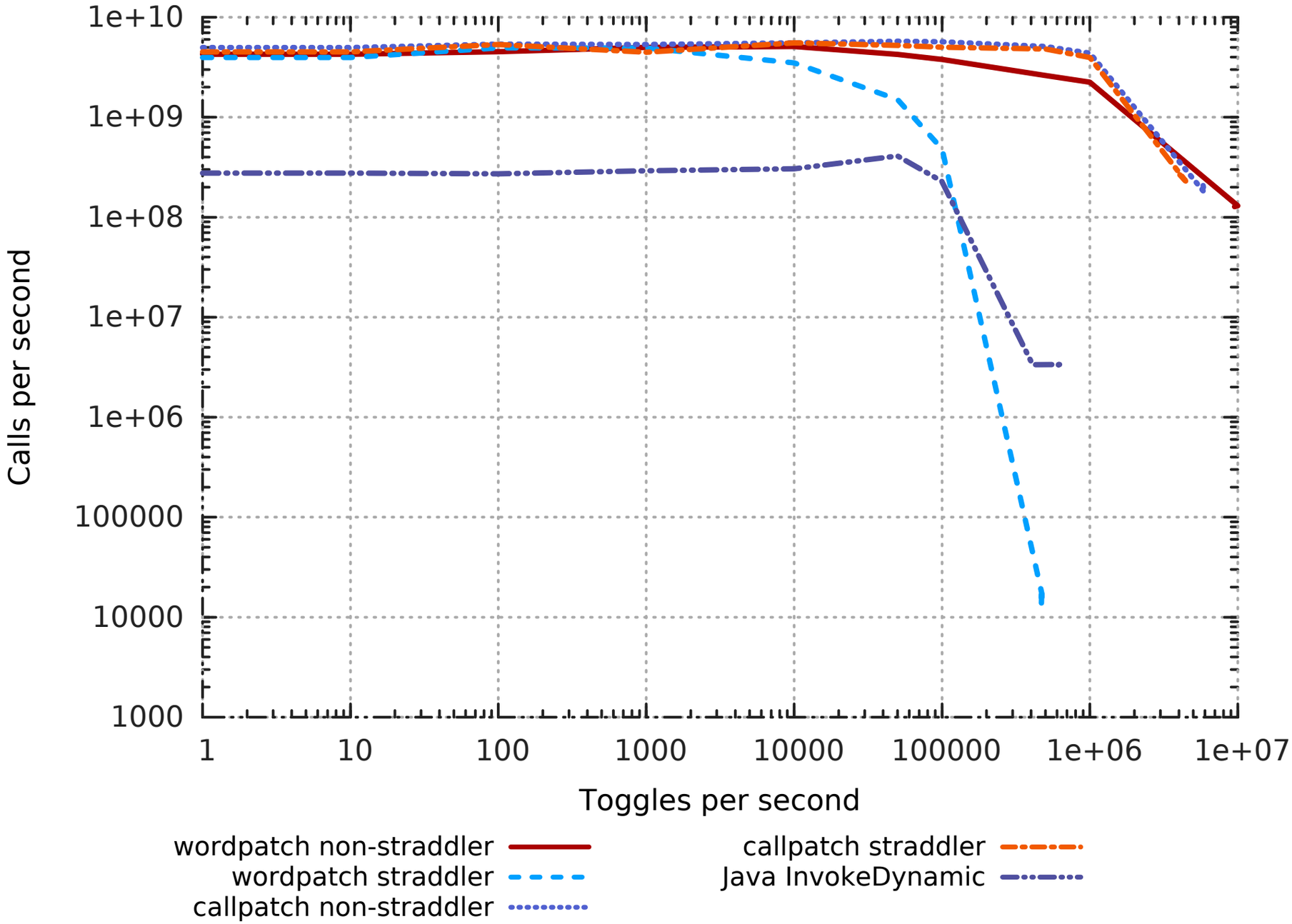
Comparison

Wordpatch



Comparison

Wordpatch



Lightweight Profiler

- Measures function call counts, average execution times
- Does sampling with self deactivated probes
- Monitor thread wakes up periodically and reactivate inactive probes.

Application Benchmarks

Benchmark	#probes	#toggles/sec	#samples/sec	slowdown%	toggle%
h264ref	638	24,015	115,526	6	0.1
bzip	162	2231	10,638	0.6	0.01
sjeng	142	14,616	73,060	3	0.06
perl	1408	42,276	211,346	11	0.2
nbody	252	8052	34,888	2	0.02
hull	164	1185	5636	0.2	0.005
blackscholes	8	730	3650	1	0.002

Application Benchmarks

Benchmark	#probes	#toggles/sec	#samples/sec	slowdown%	toggle%
h264ref	638	24,015	115,526	6	0.1
bzip	162	2231	10,638	0.6	0.01
sjeng	142	14,616	73,060	3	0.06
perl	1408	42,276	211,346	11	0.2
nbody	252	8052	34,888	2	0.02
hull	164	1185	5636	0.2	0.005
blackscholes	8	730	3650	1	0.002

Wrapping up

✓ Safe cross modification protocols on x86 for general instruction patching and call instruction patching

✓ No stop the world requirements

✓ A rapid probe toggling implementation

✓ Useful for profilers, JIT compilers, live software updaters or in other DBIs for x86 etc.

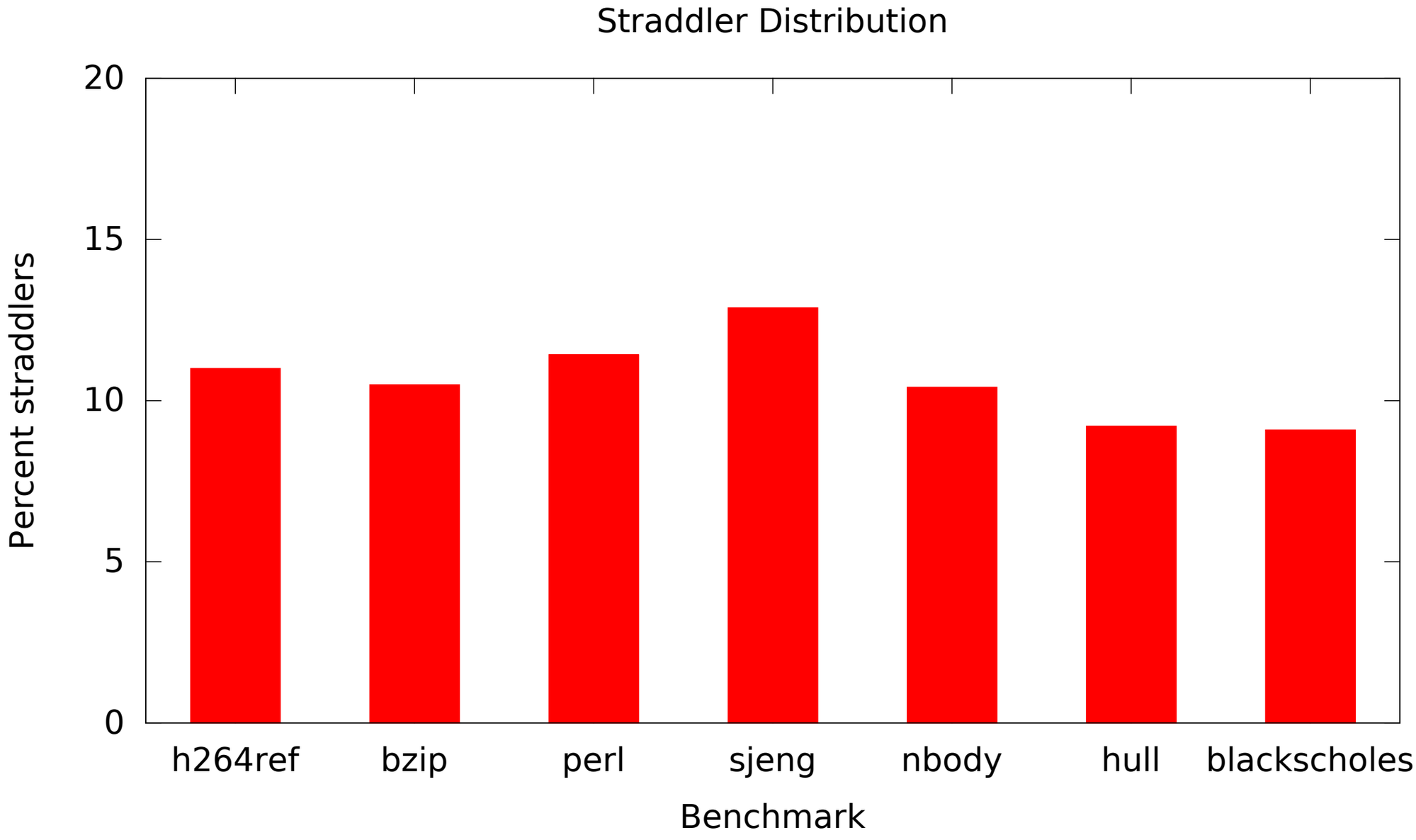
Buddhika Chamith
budkahaw@indiana.edu
github : @chamibuddhika

Safe **wordpatch**

```
//    Write an interrupt byte into the address,
void int3_lock(address)
  return write_front(address, INT3);

bool patch(address, value)
  if (not is_straddler(address))
    write(address, value);
    return true;
  else if (int3_lock(address)
    wait();
    write_back(address, value);
    wait();
    write_front(address, value);
    return true;
  else return false;
```

Straddler Distribution

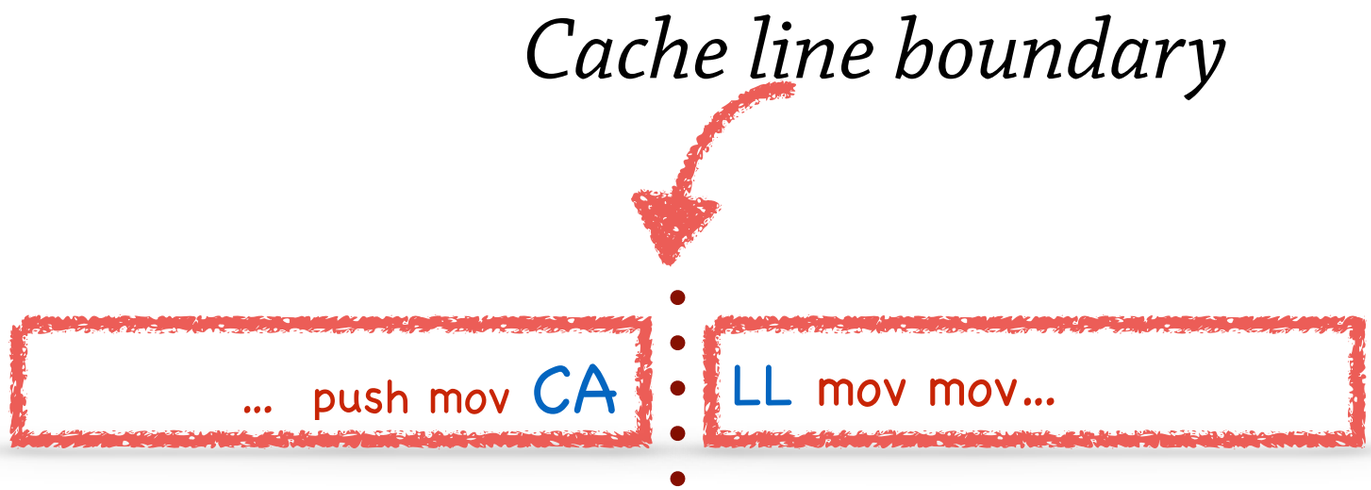


Intel Cross Modification

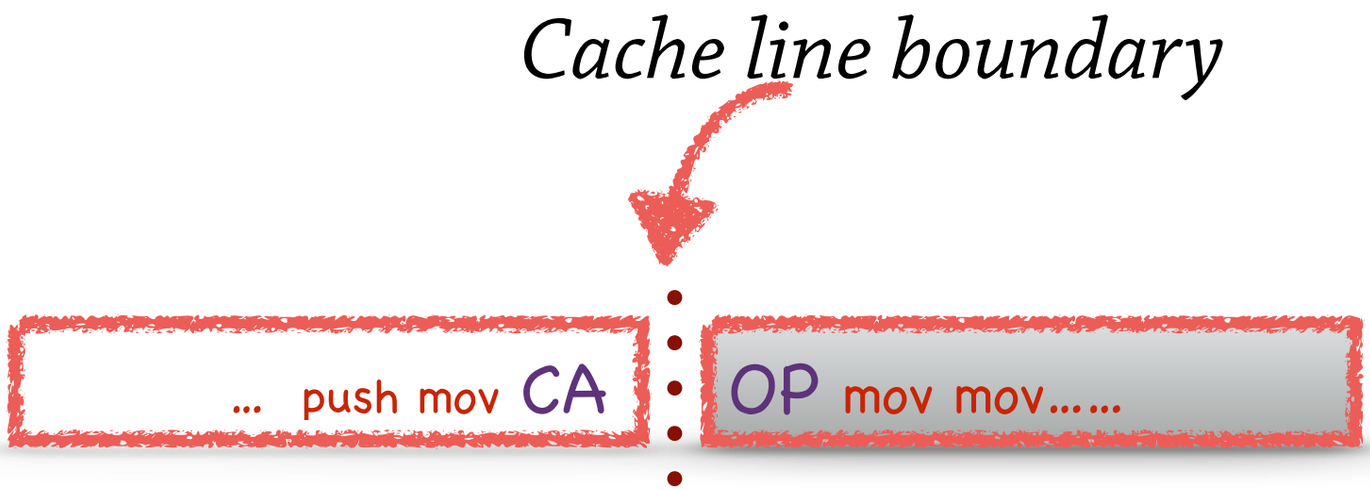
```
// -- Action of Modifying Processor --
memory_flag = 0;
Store modified code (as data) into code segment;
memory_flag = 1;

// -- Action of Executing Processor --
WHILE (memory_flag != 1)
Wait for code to update;
ELIHW;
Execute serializing instruction; // e.g: CPUID
Begin executing modified code;
```

Woes of cmpxchg



Woes of cmpxchg



INCORRECT

Latent Costs

Potential sources

- 5 byte NOOP : ~1-2 cycles
- Callpatch : Inactive state jumps
  - Short : ~2 cycles
  - Degenerate RET call : ~4 cycles

Probing Costs

Method	Activation	Deactivation	Invocation
wordpatch :			
non straddlers	300	120	35
straddlers	8850	8435	35
callpatch	432	315	35
JVM VolatileCallSite	995	995	55
Dyninst	1,929,244	995,447	320