PAPI's new Software-Defined Events for in-depth Performance Analysis

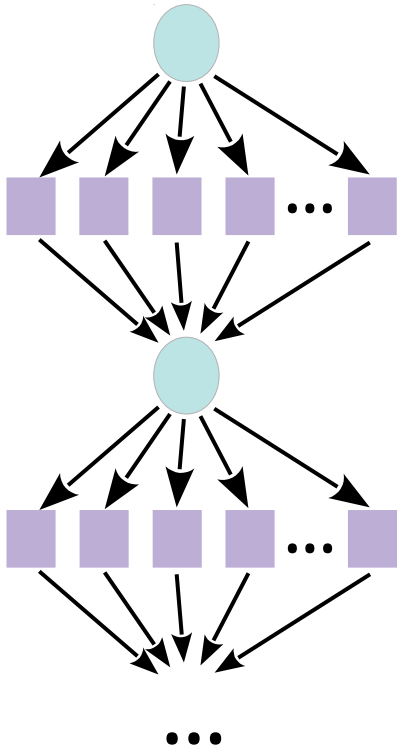
13th Parallel Tools Workshop

Anthony Danalis, Heike Jagode, Jack Dongarra

Dresden, Germany

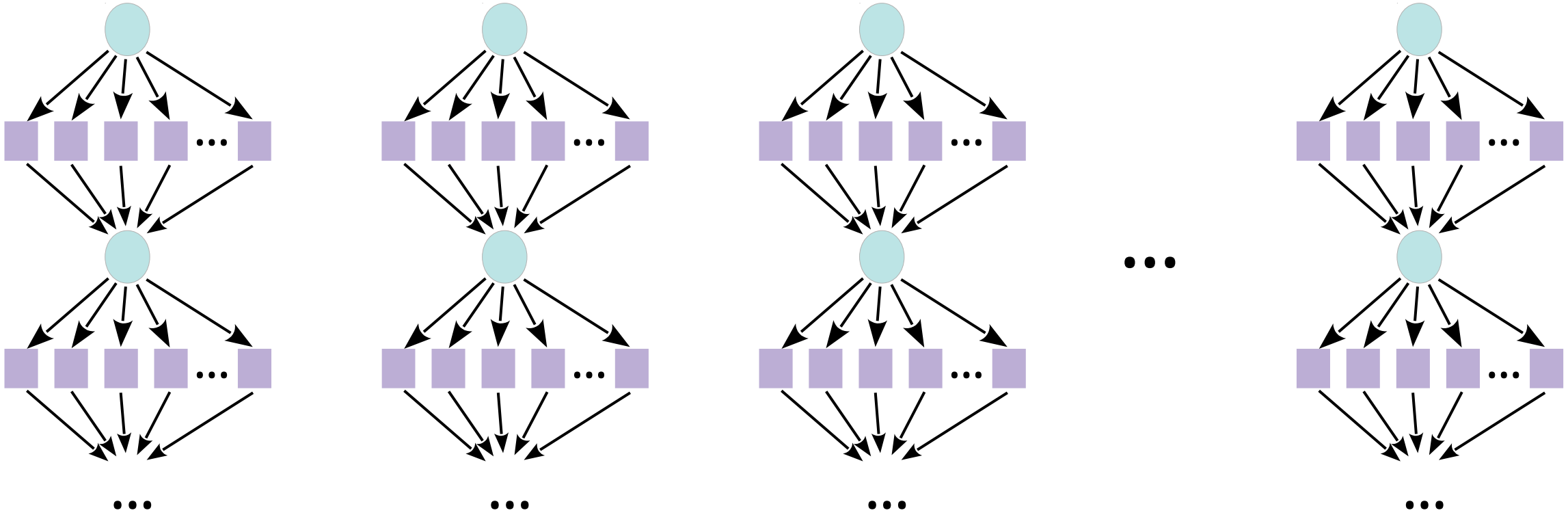
Sep. 2-3, 2019

Motivating example: Fork-Join parallelism



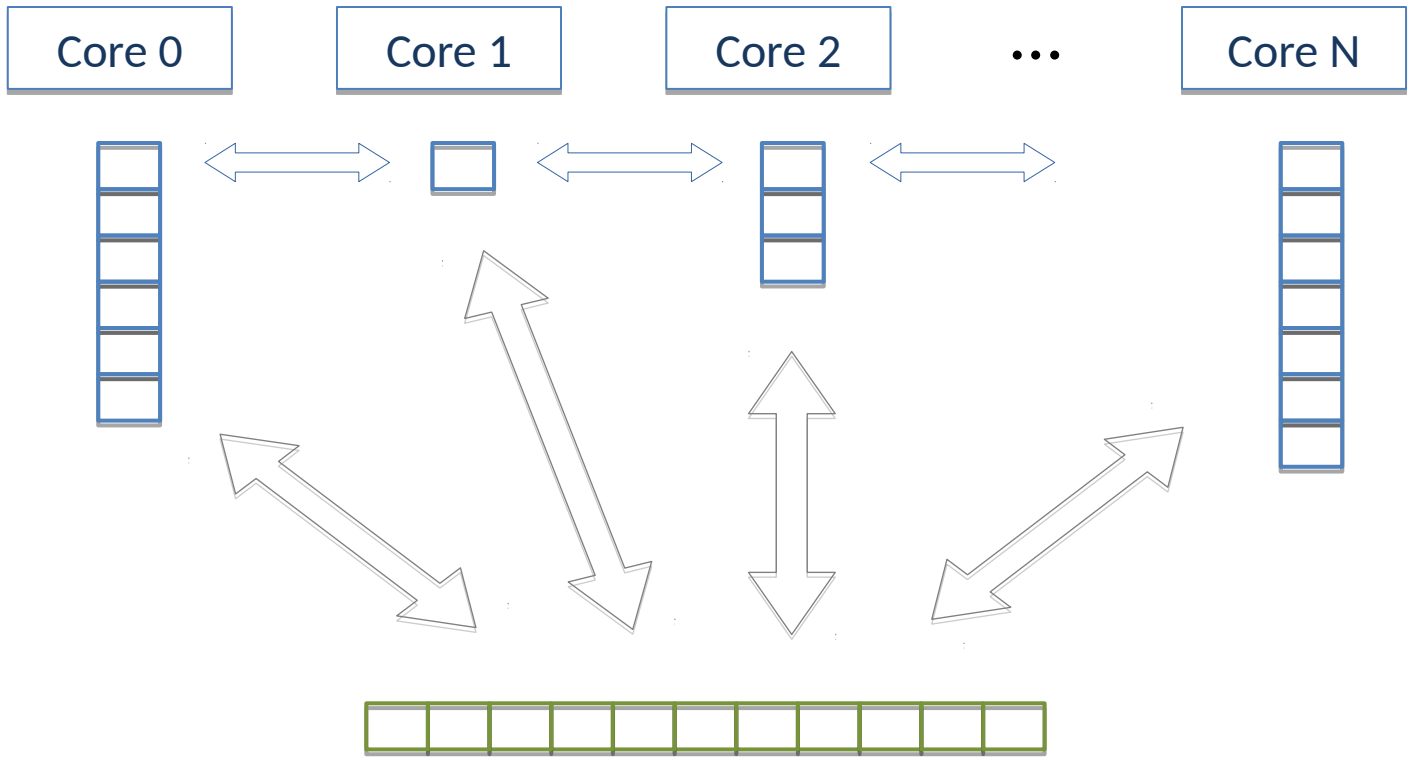
- Fork-Join chain x 20 Tasks per fork.
- Memory bound kernel, with good cache locality.

Motivating example: Fork-Join parallelism (x20)



- 20 Independent Fork-Join chains x 20 Tasks per fork.
- Memory bound kernel, with good cache locality.
- 20 Cores on testing platform.

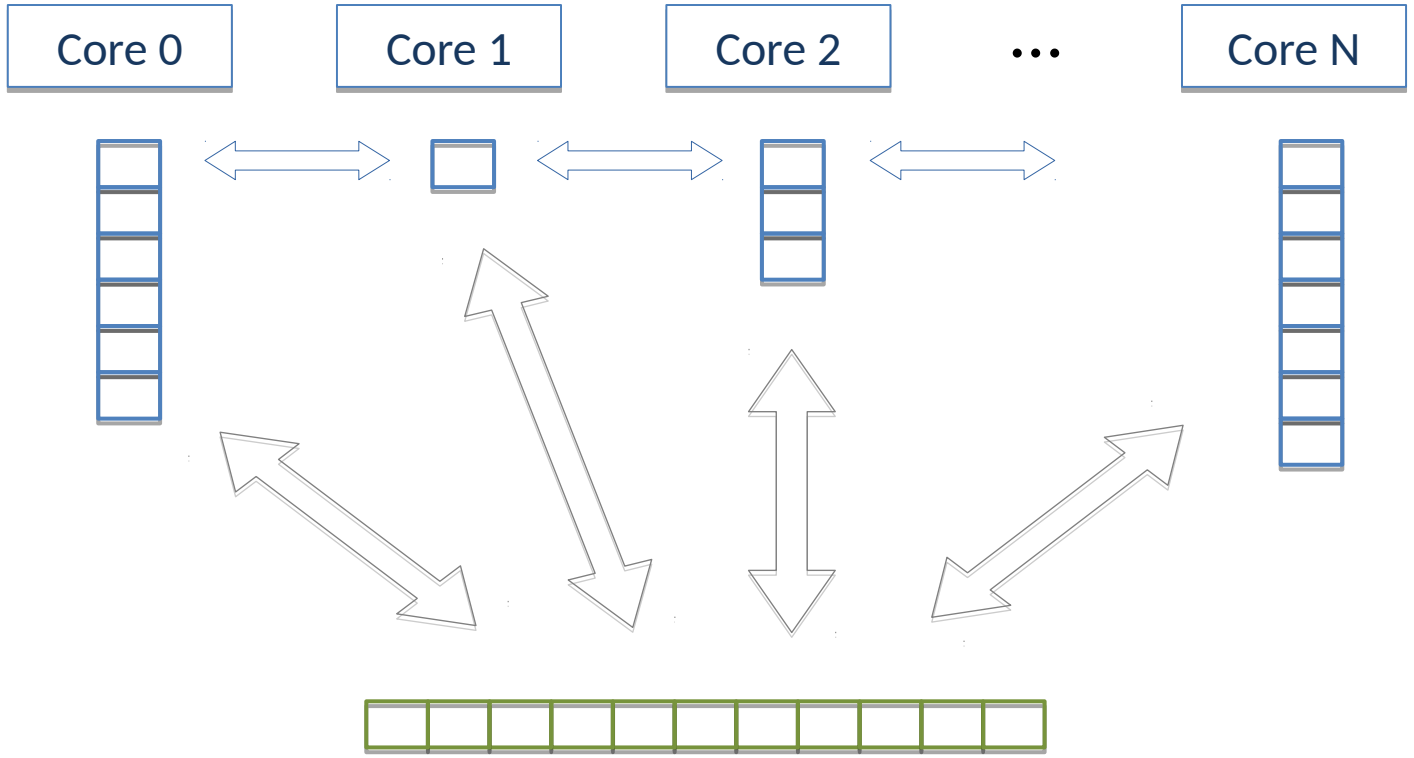
Typical task scheduling queue design



Core local queues

Shared Global queue (overflow)

Typical task scheduling queue design

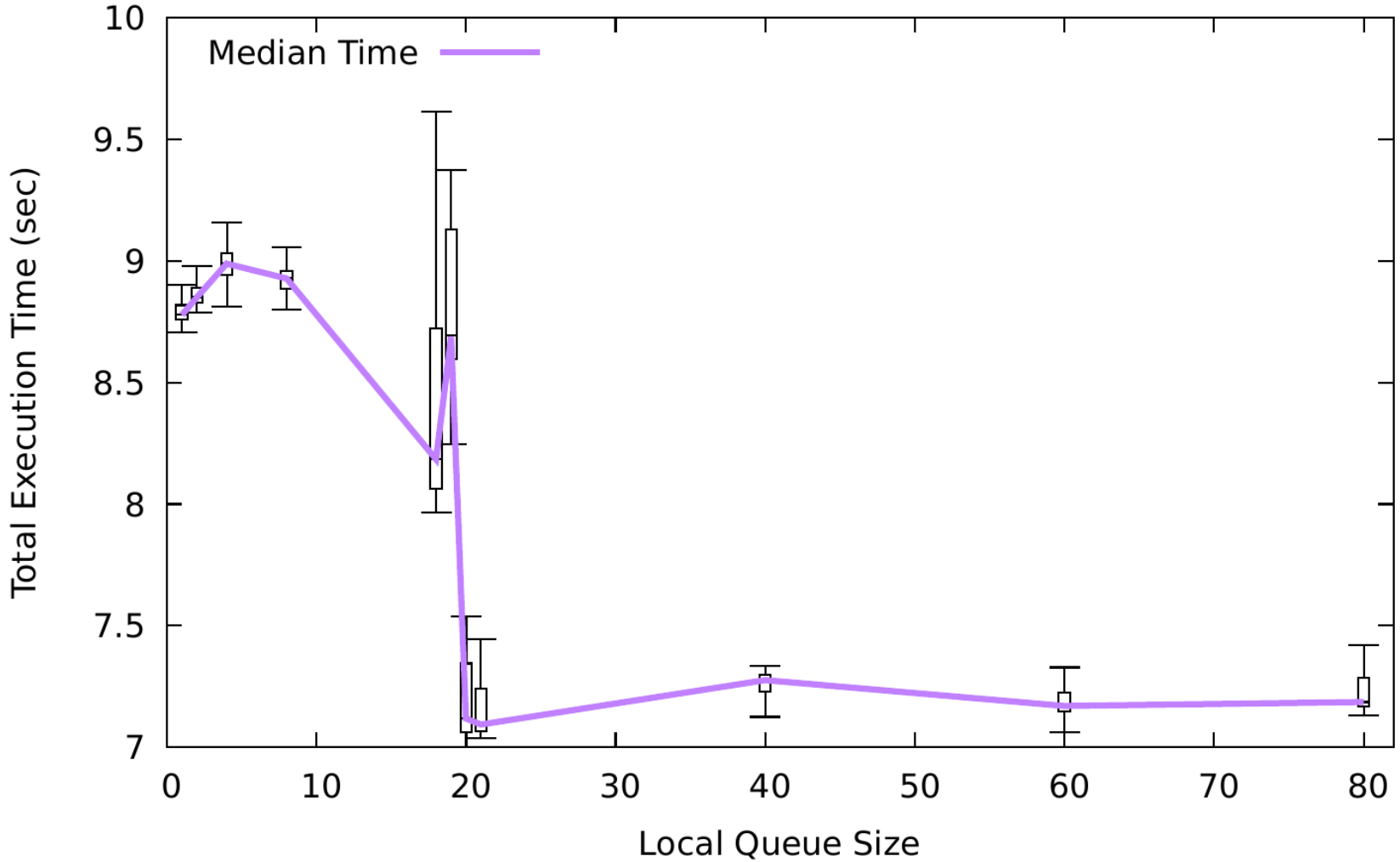


Core local queues

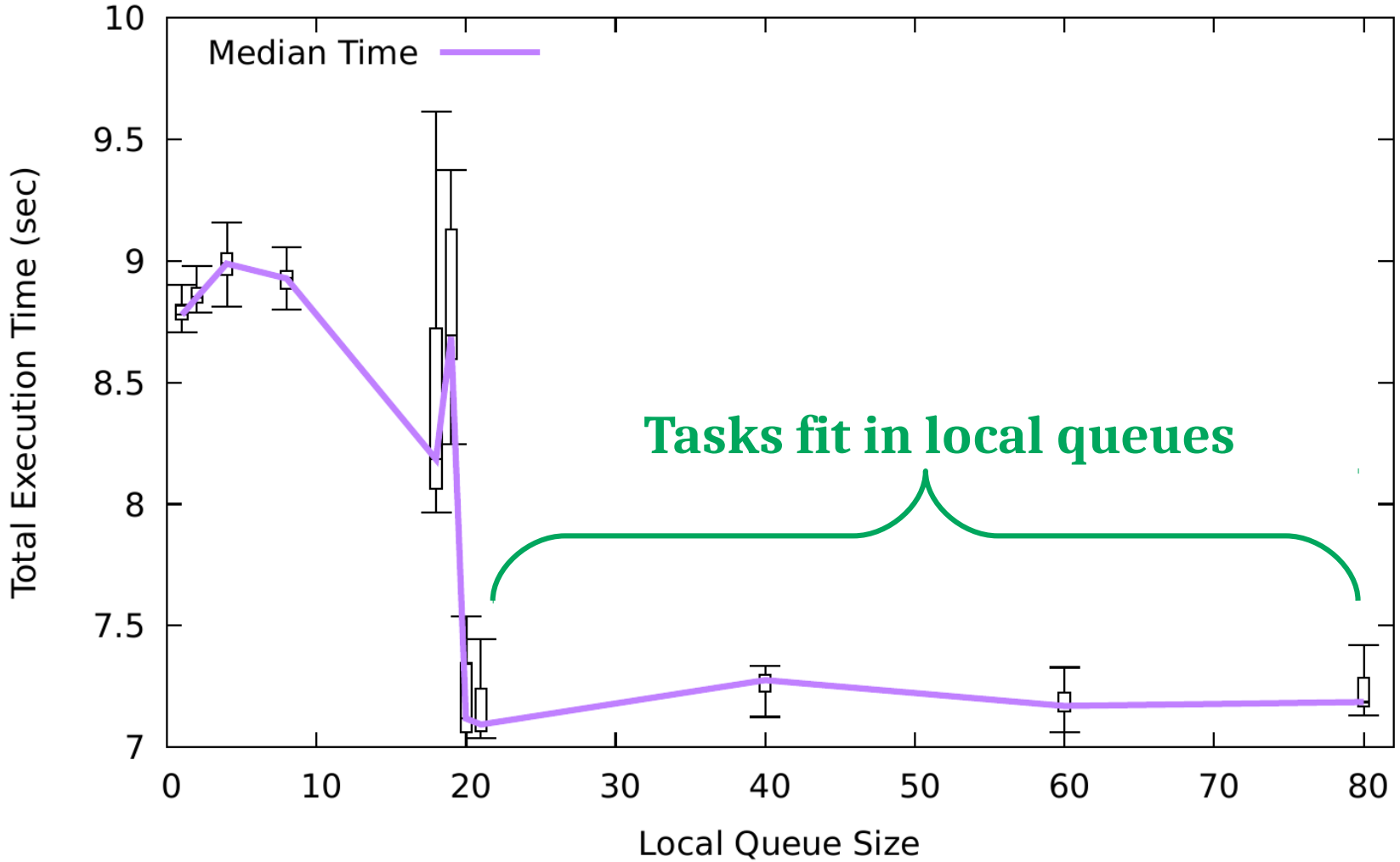
Shared Global queue (overflow)

Thread Local Queues => High Locality
Overflow & Work Stealing => Load Balance

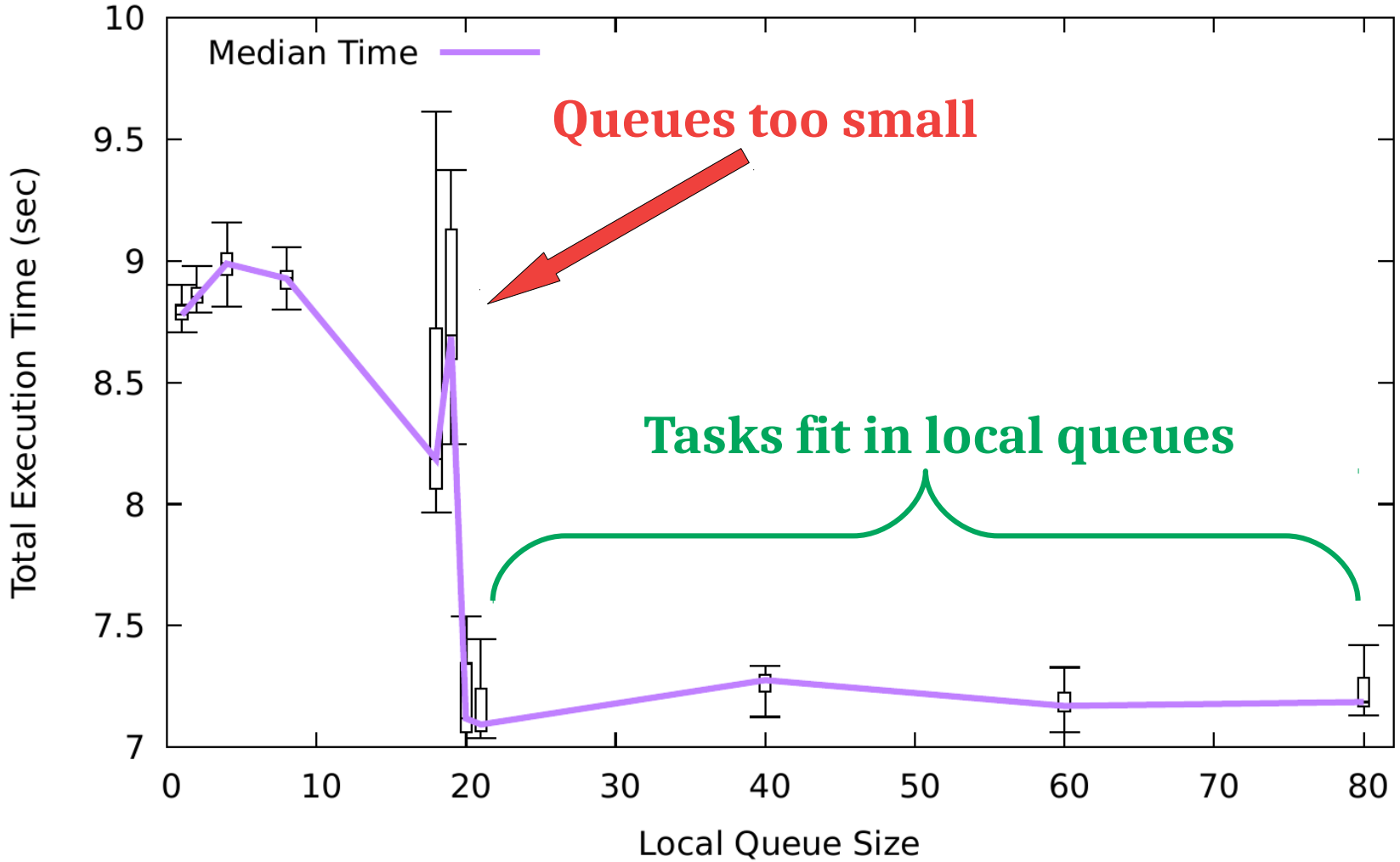
Execution time vs Local Queue Length



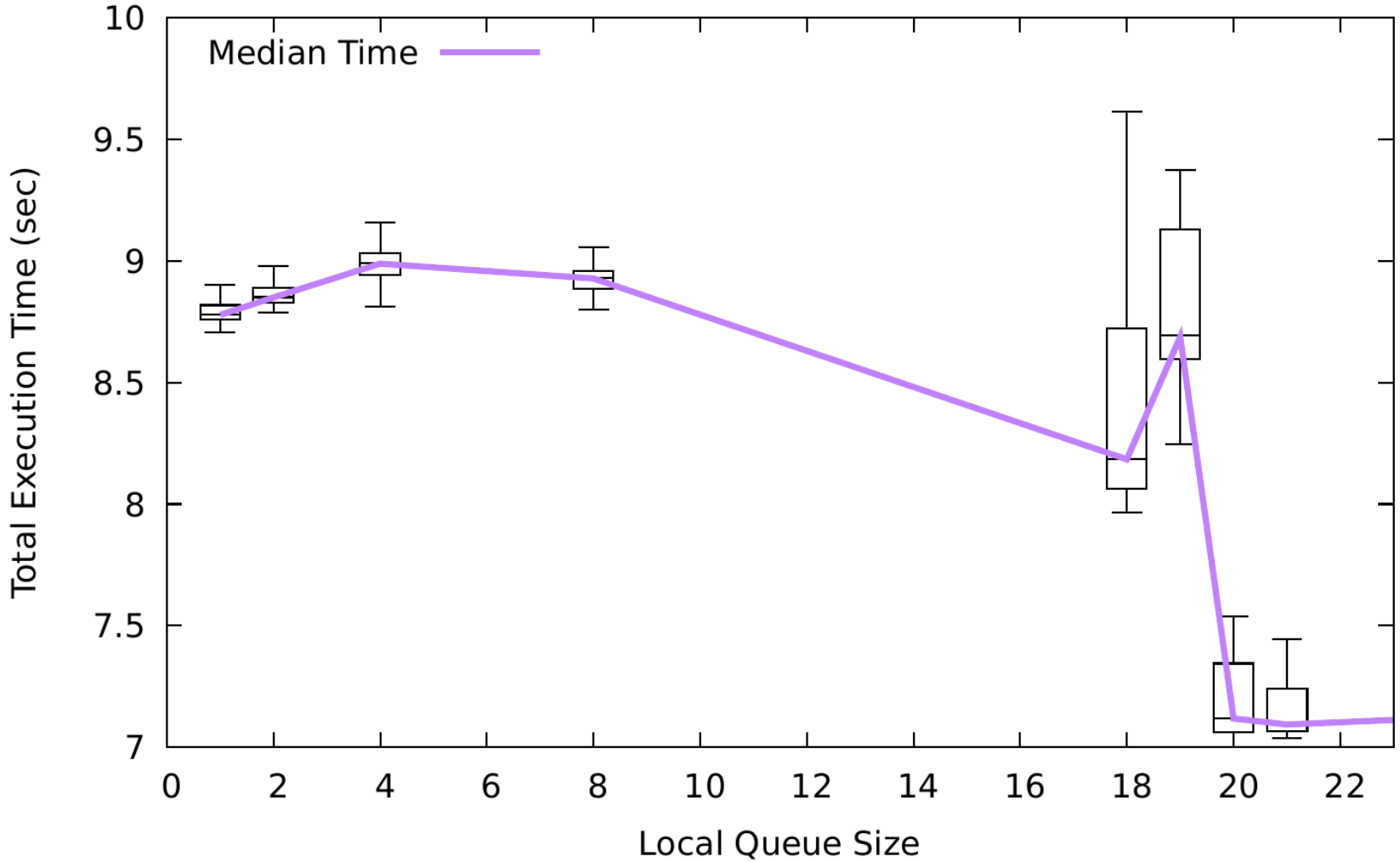
Execution time vs Local Queue Length



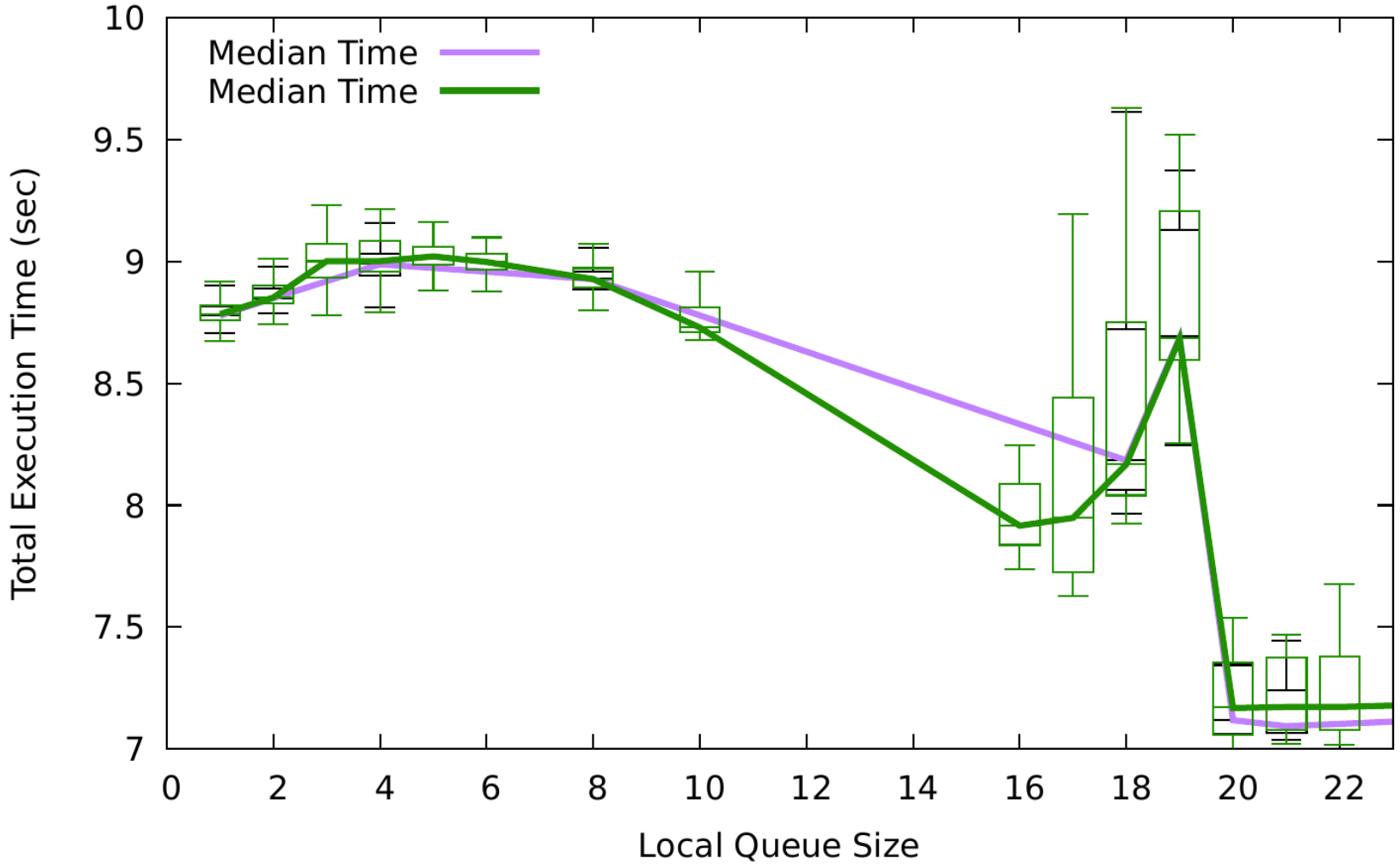
Execution time vs Local Queue Length



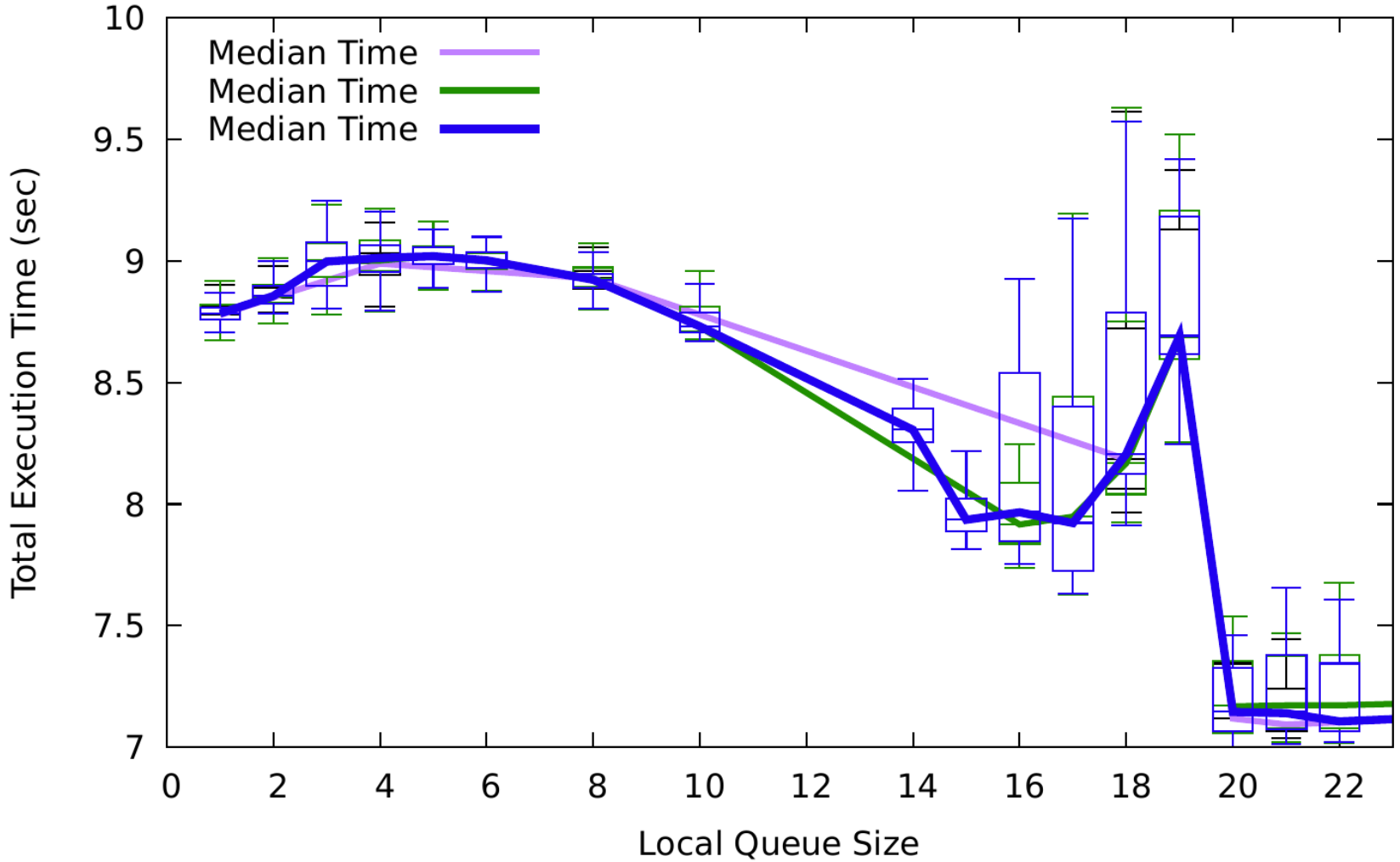
Execution time vs Local Queue Length (zoom)



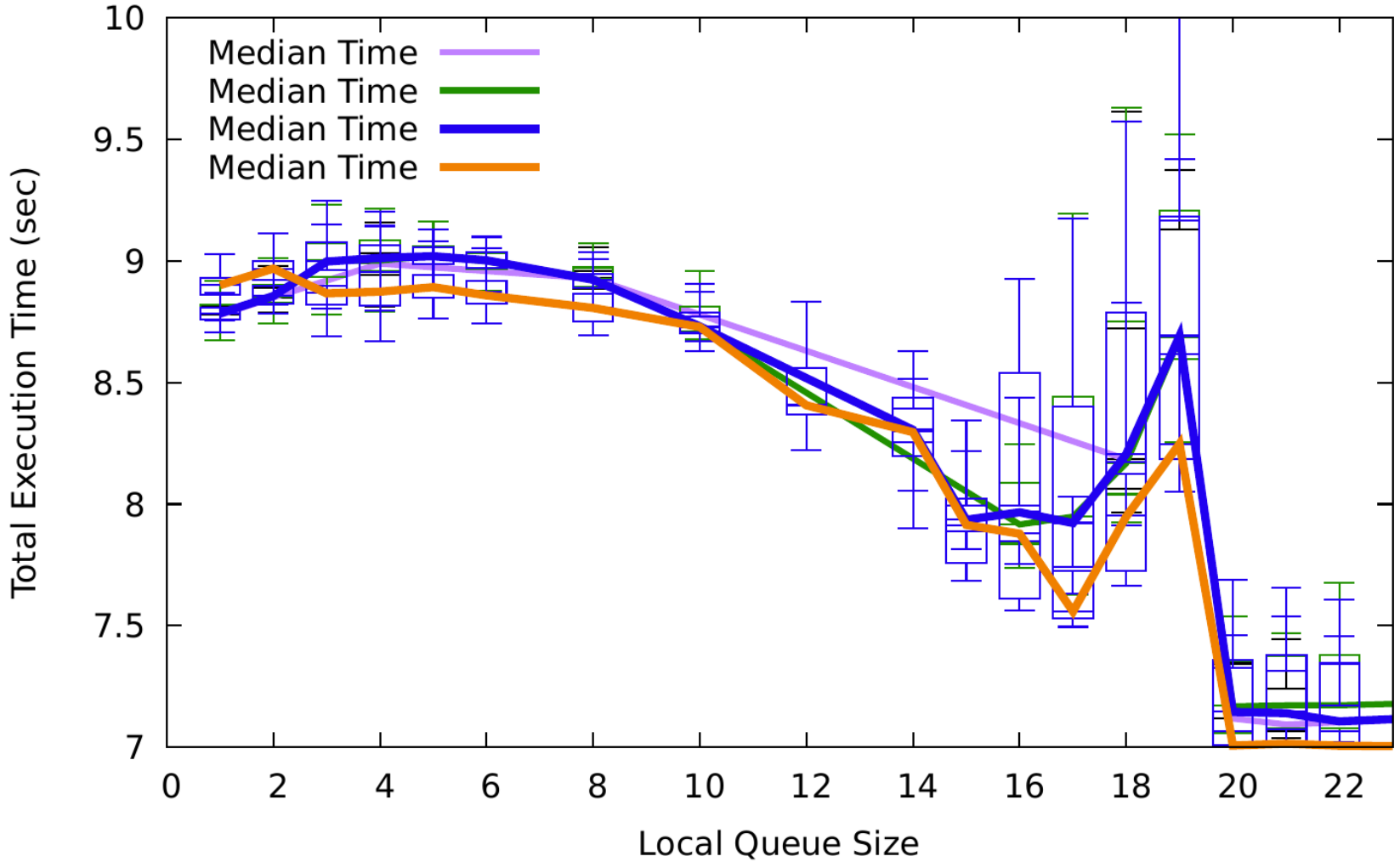
Execution time vs Local Queue Length (zoom 2)



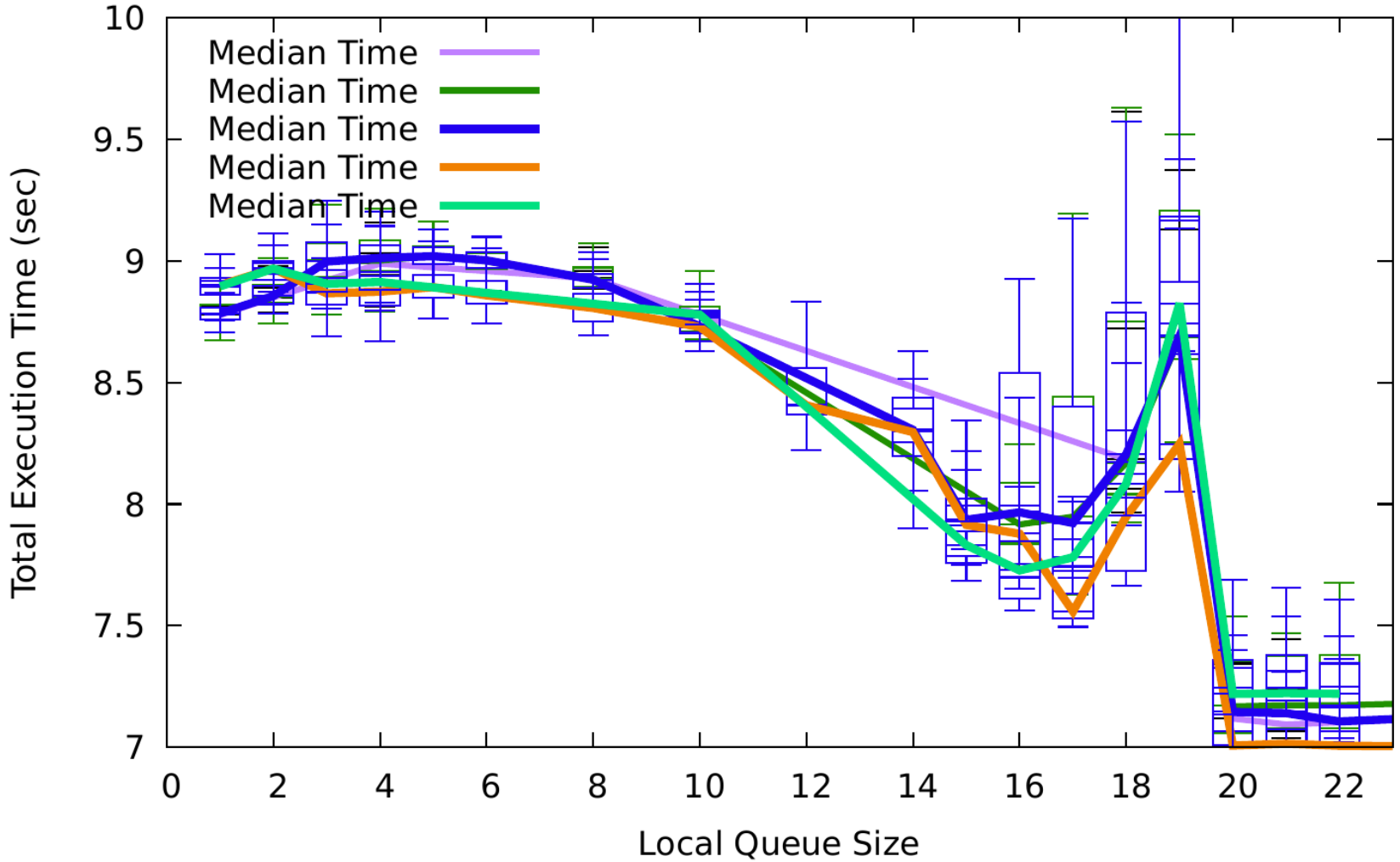
Execution time vs Local Queue Length (zoom 3)



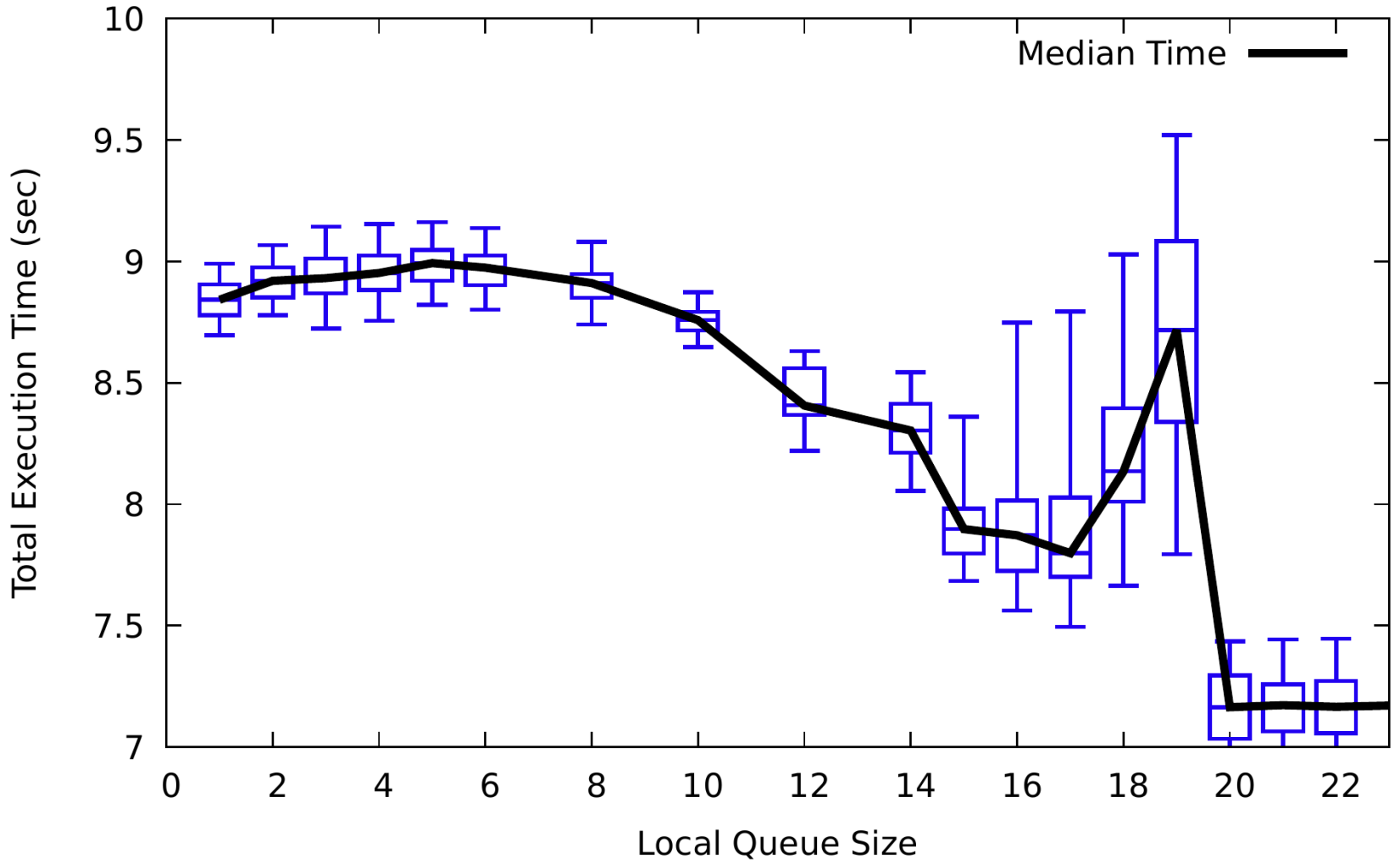
Execution time vs Local Queue Length (zoom 4)



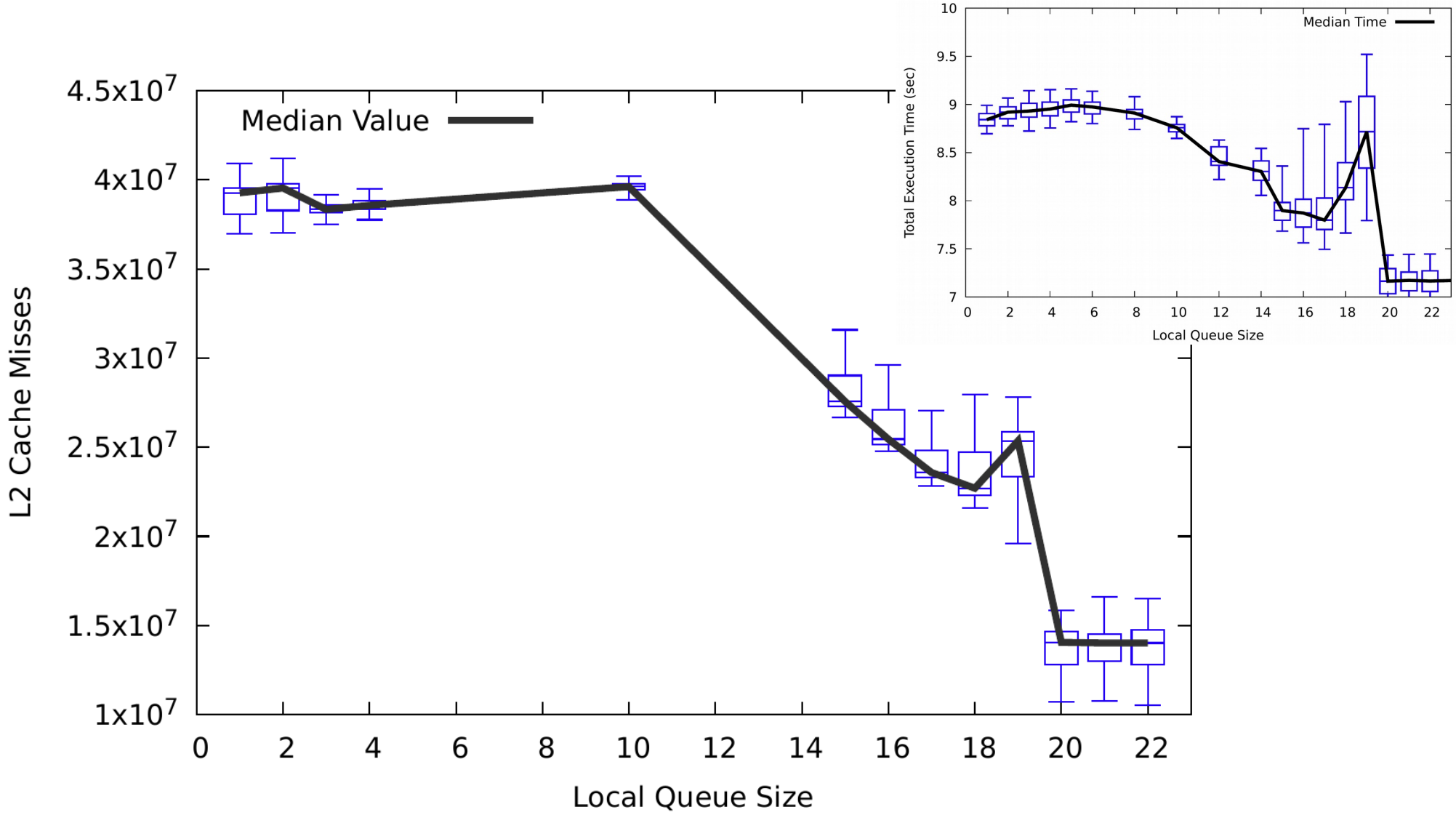
Execution time vs Local Queue Length (zoom 5)



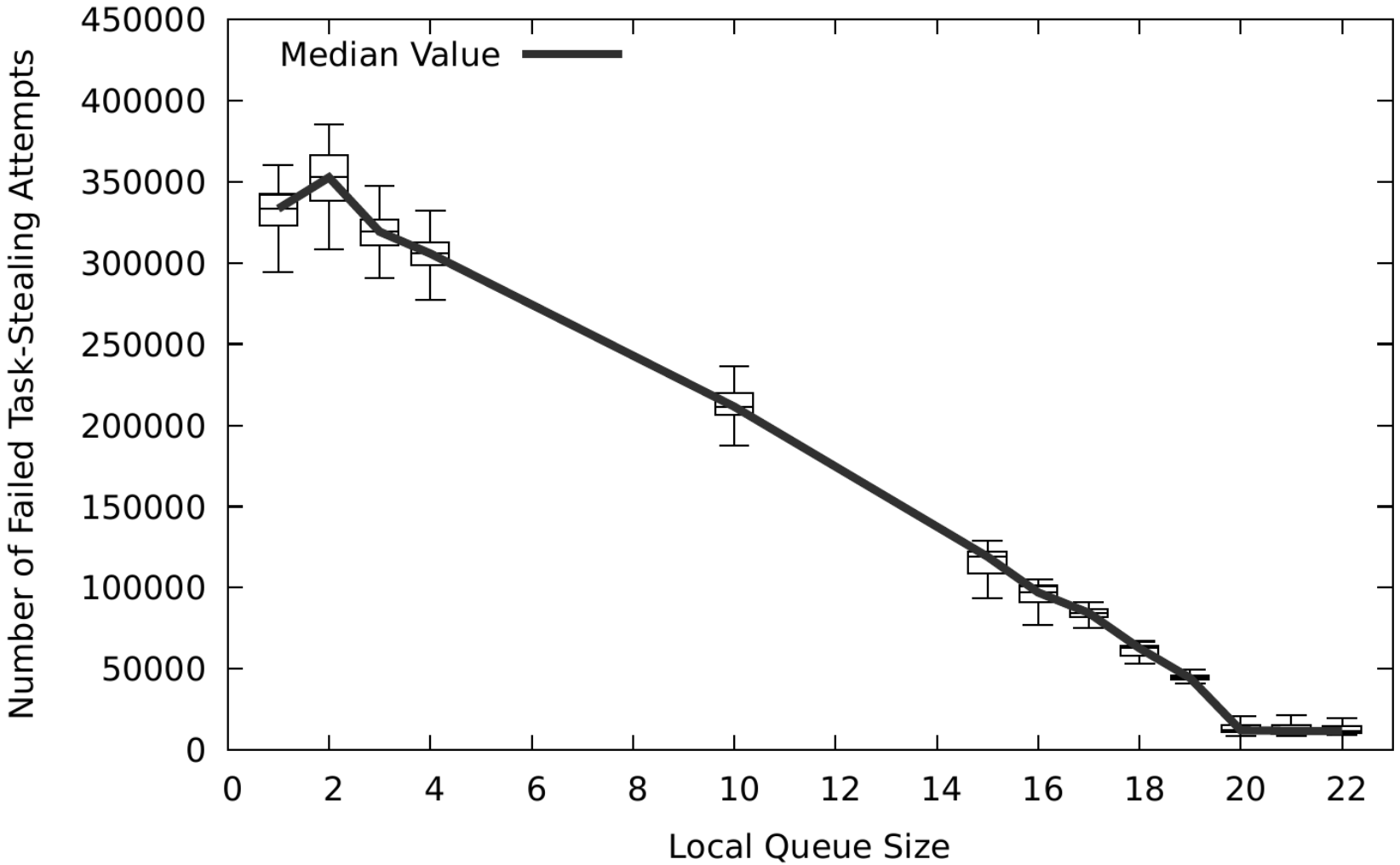
Execution time vs Local Queue Length (combined)



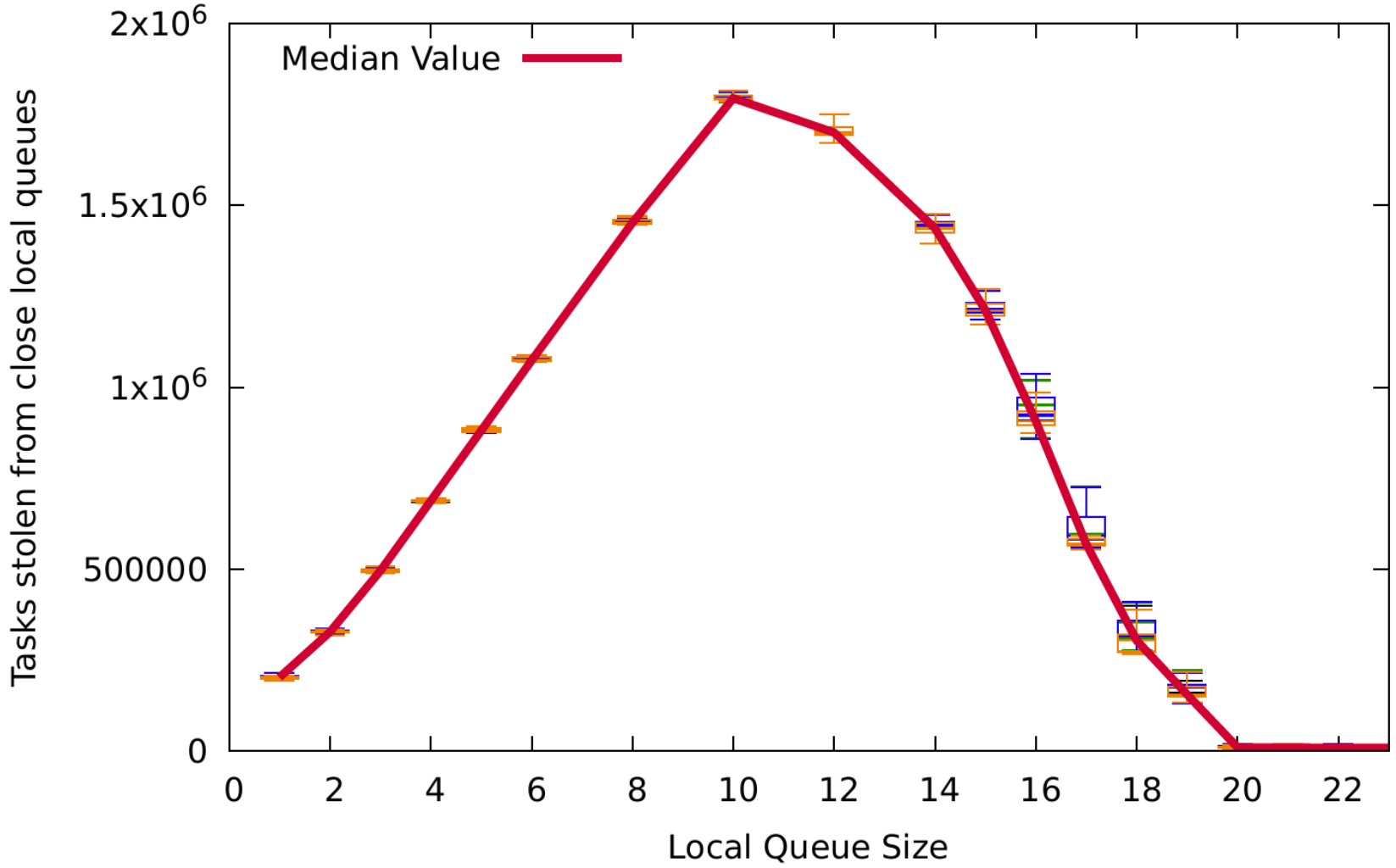
L2 Cache Misses (L3 show same pattern)



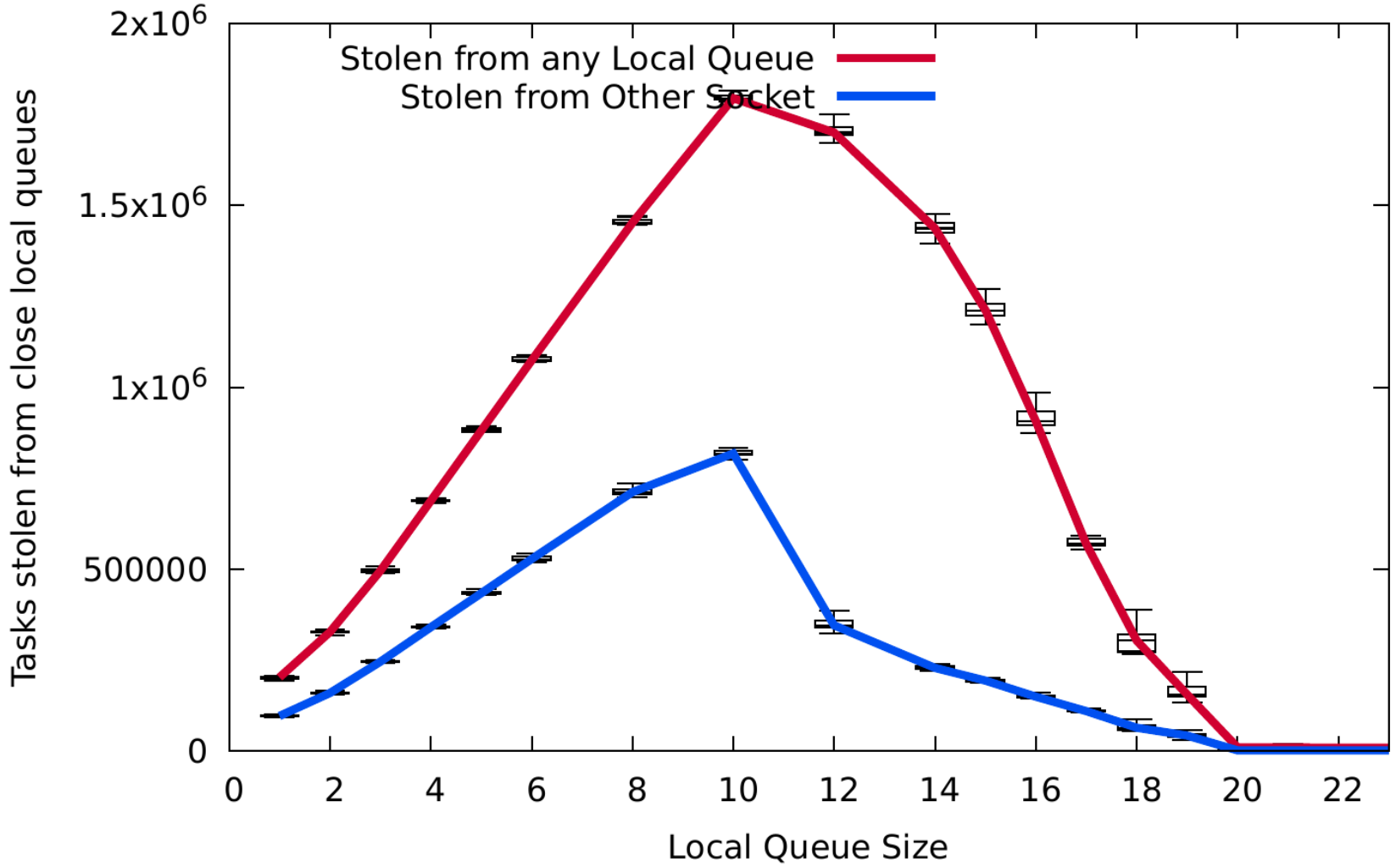
Failed Stealing Attempts



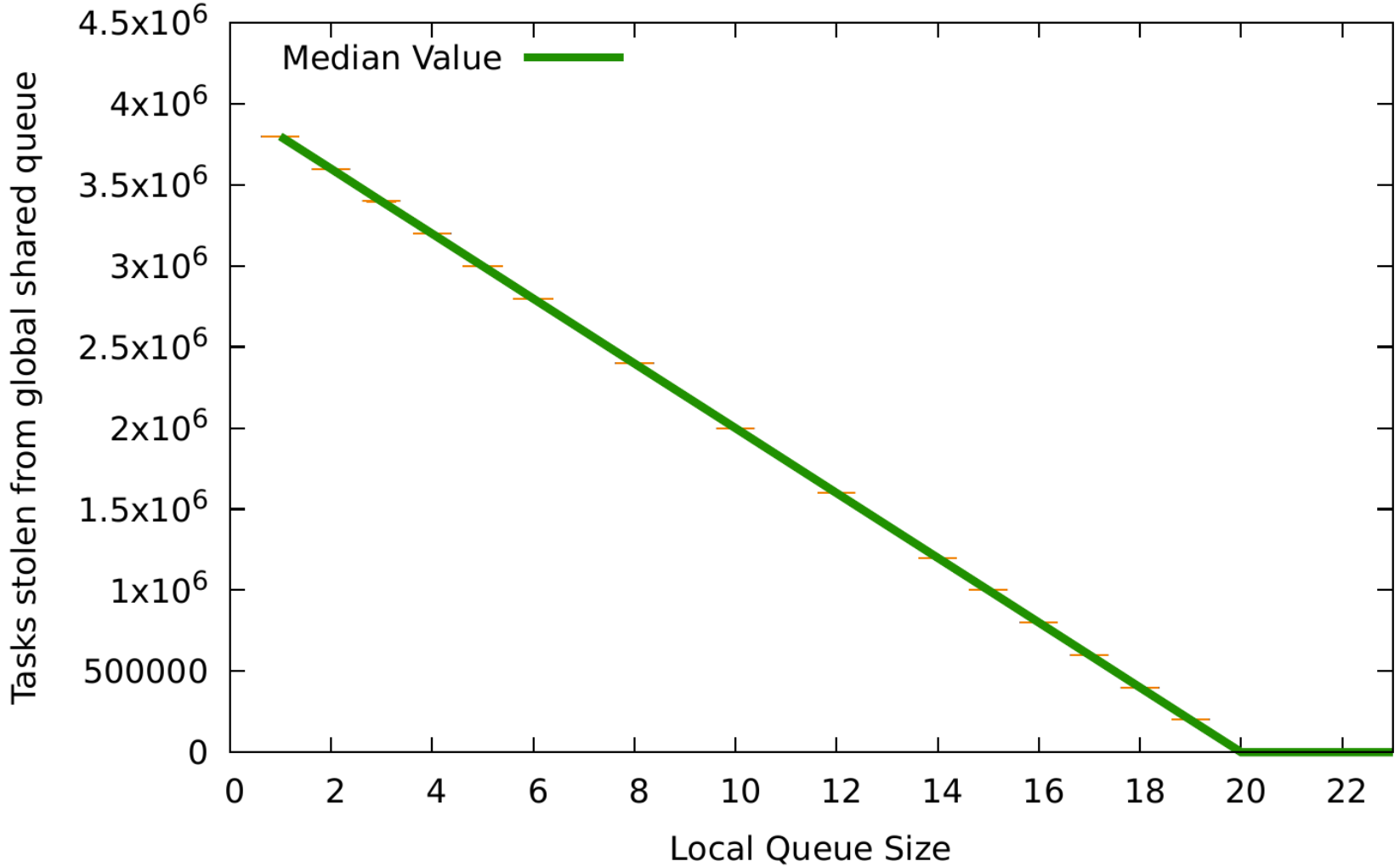
Successful Close Stealing



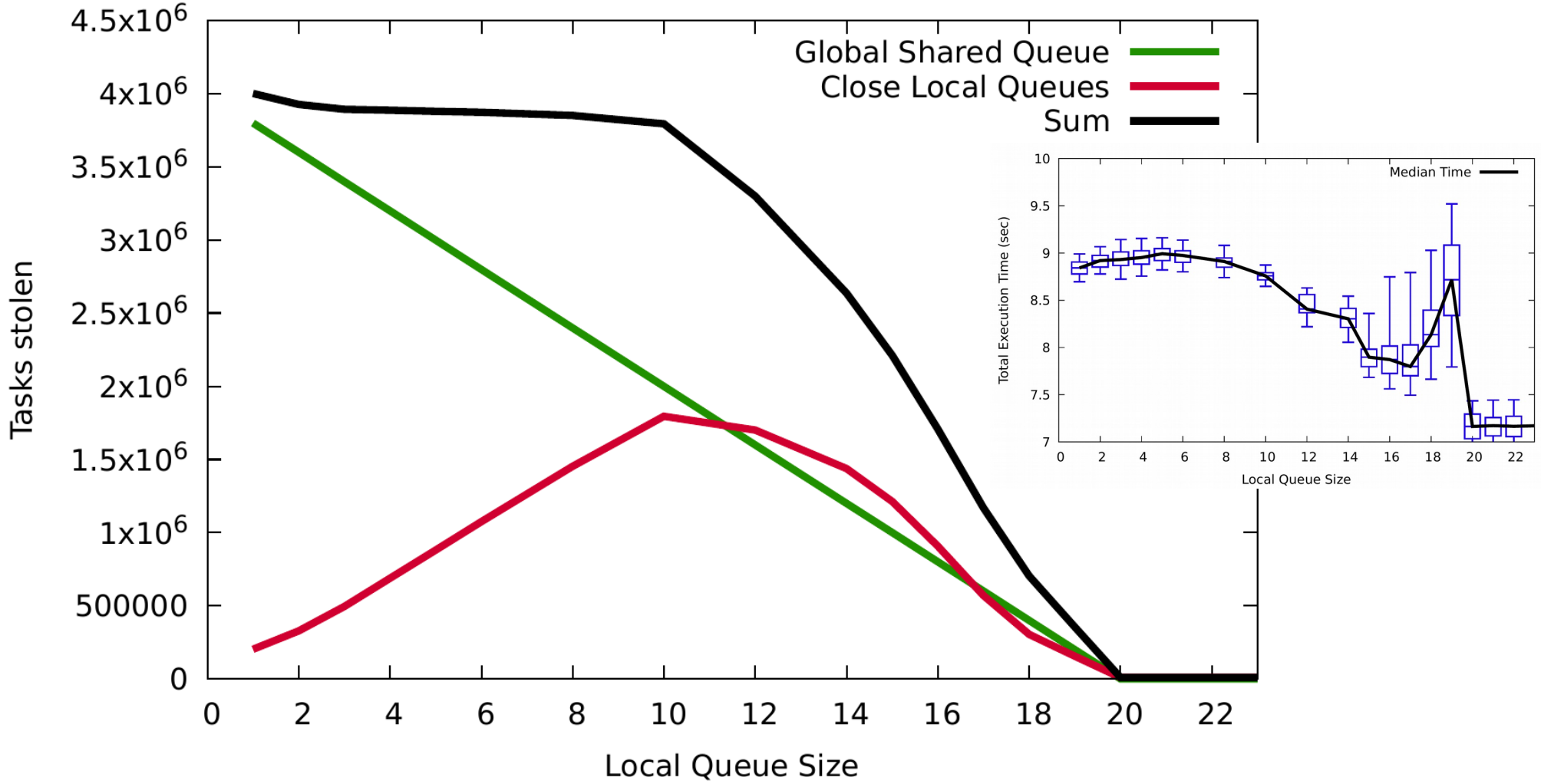
Successful Close & Far Stealing



Successful Shared Queue Stealing



Successful Local + Shared Queue Stealing



Unanswered questions

Q: So, what causes the bump?

Q: How did you measure all these things?

Unanswered questions

Q: So, what causes the bump?

A: I don’t know!

Q: How did you measure all these things?

Unanswered questions

Q: So, what causes the bump?

A: I don’t know!

Q: How did you measure all these things?

A: I am glad you asked…

Unanswered questions

Q: So, what causes the bump?

A: I don’t know!

Q: How did you measure all these things?

A: I am glad you asked… with PAPI SDEs of course!

Facts about PAPI Software Defined Events

- New measurement possibilities:

  Tasks stolen, matrix residuals, partial results reached, arguments passed to functions

Facts about PAPI Software Defined Events

- New measurement possibilities:

  Tasks stolen, matrix residuals, partial results reached, arguments passed to functions

- Any tool can read PAPI SDEs:

  SDEs from a library can be read with PAPI_start()/PAPI_stop()/PAPI_read().

Facts about PAPI Software Defined Events

- New measurement possibilities:

  Tasks stolen, matrix residuals, partial results reached, arguments passed to functions

- Any tool can read PAPI SDEs:

  SDEs from a library can be read with PAPI_start()/PAPI_stop()/PAPI_read().

- Low overhead:

  Performance critical codes can implement SDEs with zero overhead

Facts about PAPI Software Defined Events

- New measurement possibilities:

  Tasks stolen, matrix residuals, partial results reached, arguments passed to functions

- Any tool can read PAPI SDEs:

  SDEs from a library can be read with PAPI_start()/PAPI_stop()/PAPI_read().

- Low overhead:

  Performance critical codes can implement SDEs with zero overhead

- Easy to use, with rich feature set:

  Pull-mode & push-mode, read-write counters, sampling/overflowing, counters, groups, recordings, statistics, thread safety, custom callbacks

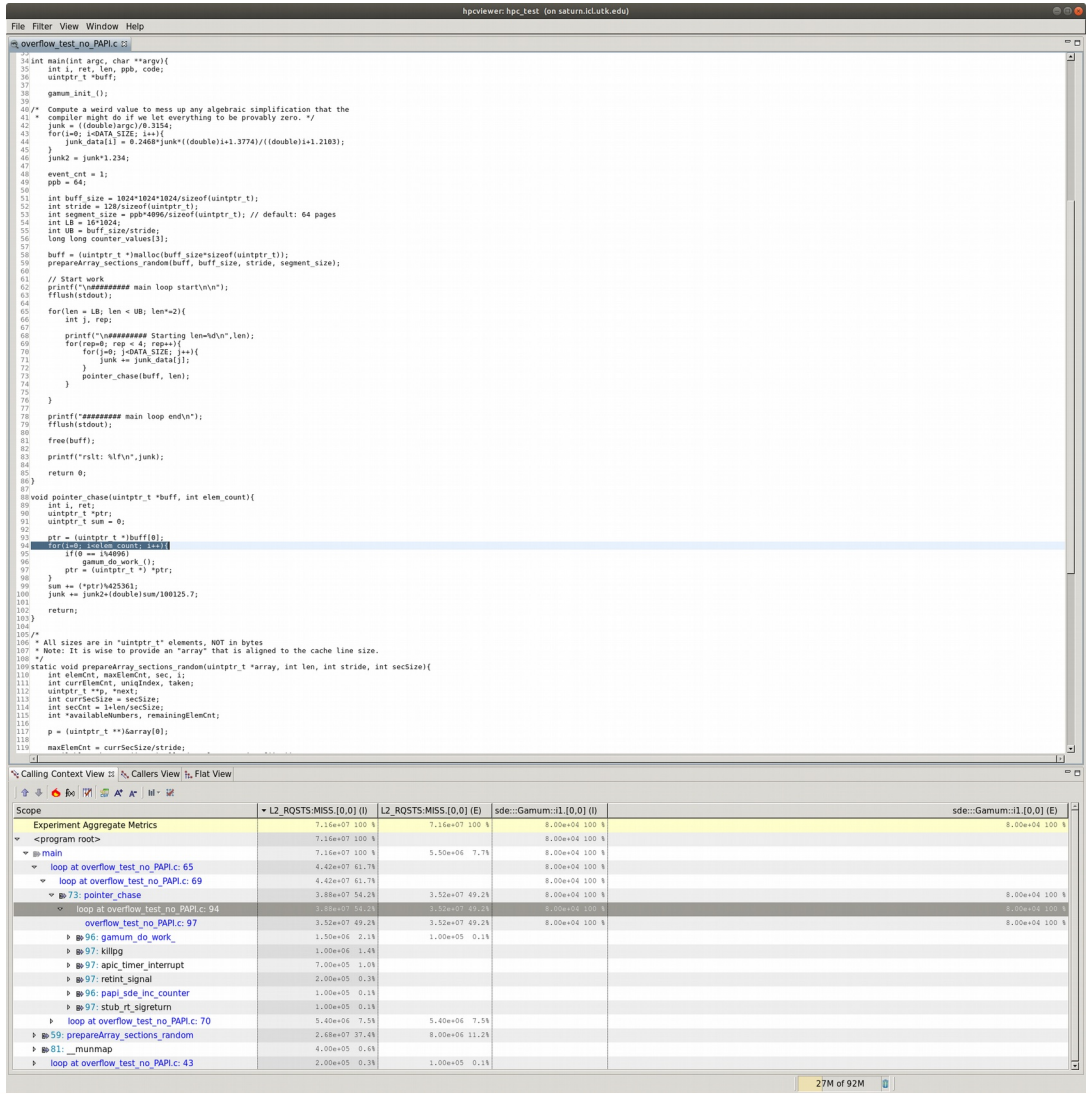
What was missing from existing infrastructure?

Events that occurred inside the software stack

There is no standardized way for a software layer to export information about its behavior such that other, independently developed, software layers can read it.

HPC Application	Quantum Chemistry Method
Math library	Distributed Factorization
Task runtime	Data Dependency
MPI	One Sided Communication
Libibverbs	RDMA completion

Stock HPCToolkit



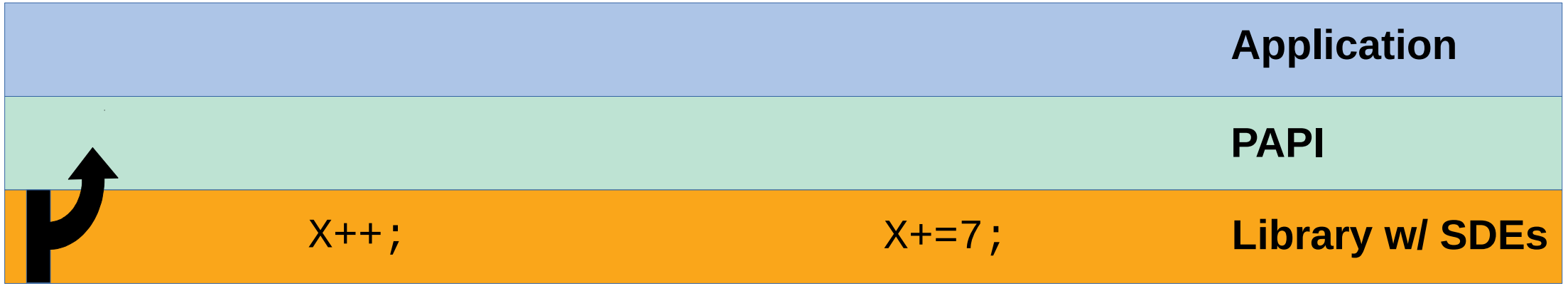
Stock HPCToolkit (zoom)

```
94      for(i=0; i<elem_count; i++){
95          if(0 == i%4096)
96              gamum_do_work_();
97          ptr = (uintptr_t *) *ptr;
98      }
99      sum += (*ptr)%425361;
100     junk += junk2+(double)sum/100125.7;
101
102     return;
103 }
104
105 /*
106  * All sizes are in "uintptr_t" elements, NOT in bytes
107  * Note: It is wise to provide an "array" that is aligned to the cache line size.
108  */
109 static void prepareArray_sections_random(uintptr_t *array, int len, int stride, int secSize){
110     int elemCnt, maxElemCnt, sec, i;
111     int currElemCnt, uniqIndex, taken;
112     uintptr_t **p, *next;
113     int currSecSize = secSize;
114     int secCnt = 1+len/secSize;
115     int *availableNumbers, remainingElemCnt;
116
117     p = (uintptr_t **)&array[0];
118
119     maxElemCnt = currSecSize/stride;
```

Calling Context View | Callers View | Flat View

Scope	L2_RQSTS:MISS.[0,0] (I)	L2_RQSTS:MISS.[0,0] (E)	sde:::Gamum::i1.[0,0] (I)
Experiment Aggregate Metrics	7.16e+07 100 %	7.16e+07 100 %	8.00e+04 100 %
<program root>	7.16e+07 100 %		8.00e+04 100 %
main	7.16e+07 100 %	5.50e+06 7.7%	8.00e+04 100 %
loop at overflow_test_no_PAPI.c: 65	4.42e+07 61.7%		8.00e+04 100 %
loop at overflow_test_no_PAPI.c: 69	4.42e+07 61.7%		8.00e+04 100 %
73: pointer_chase	3.88e+07 54.2%	3.52e+07 49.2%	8.00e+04 100 %
loop at overflow_test_no_PAPI.c: 94	3.88e+07 54.2%	3.52e+07 49.2%	8.00e+04 100 %

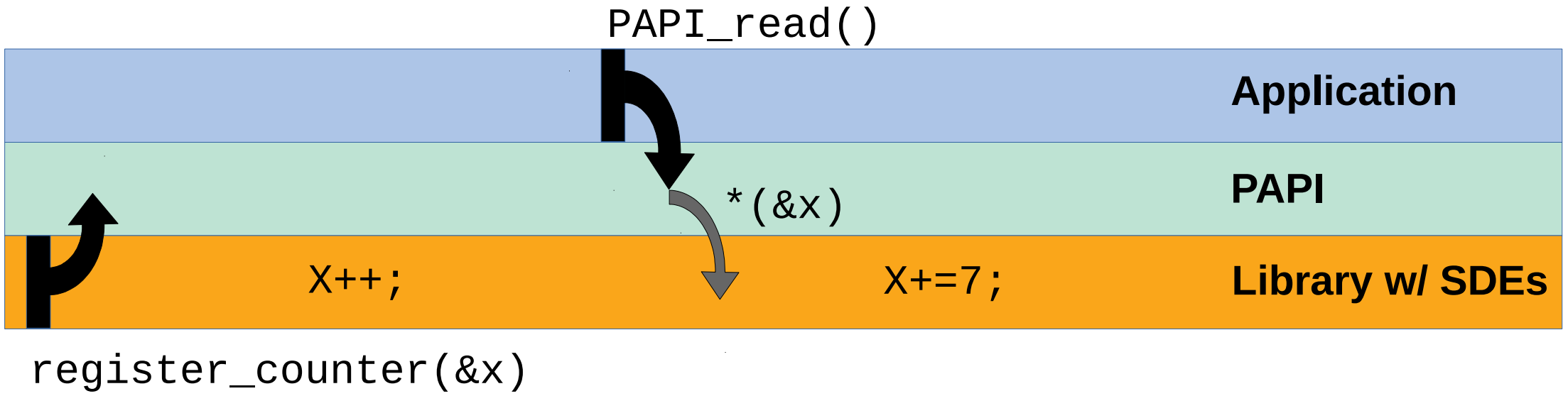
Pull mode: Low overhead (down to zero)




```
register_counter(&x)
```

Pull mode: Low overhead (down to zero)

The application reads whenever it deems necessary.



Simplest SDE code (library side)

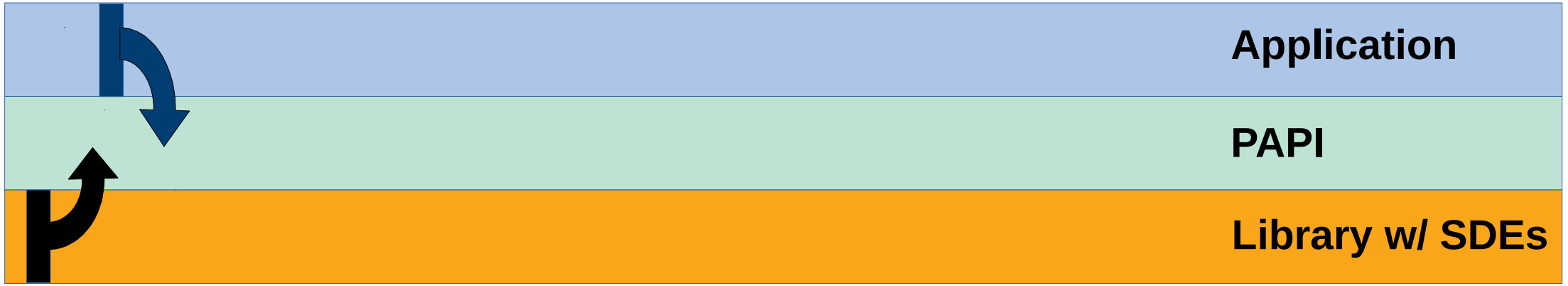
```
static long long local_var;

void small_test_init( void ){
    local_var = 0;
    papi_handle_t *handle = papi_sde_init("TEST");
    papi_sde_register_counter( handle, "Evnt",
                               PAPI_SDE_RO|PAPI_SDE_DELTA,
                               PAPI_SDE_long_long,
                               &local_var );
    ...
}
```

SDE code for registering a callback function

```
sometype_t *data;

void small_test_init( void ){
    data = ...
    papi_handle_t *handle = papi_sde_init(”TEST");
    papi_sde_register_fp_counter(handle, "Evnt",
                                 PAPI_SDE_RO|PAPI_SDE_DELTA,
                                 PAPI_SDE_long_long,
                                 accessor, data);
    ...
}
```

Push mode: Determinism and Precision

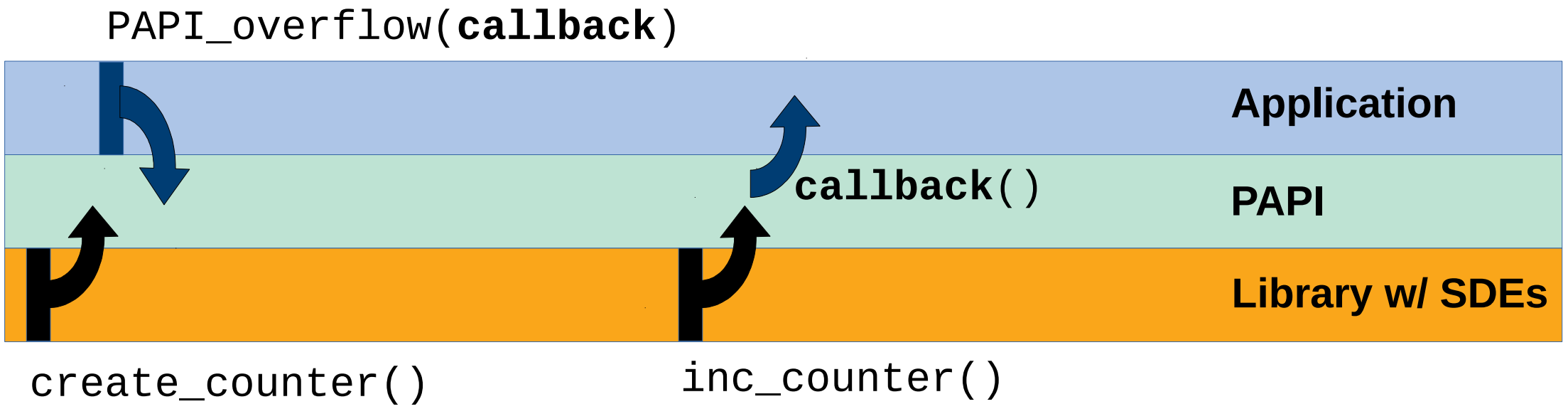
`PAPI_overflow(`**`callback`**`)`



`create_counter()`

Push mode: Determinism and Precision

The library notifies the application when something happens.



SDE code for creating a counter (push mode)

```
void *counter_handle;

void small_test_init( void ){
    papi_handle_t *handle = papi_sde_init(”TEST");
    papi_sde_create_counter(handle, "Evnt",
                            PAPI_SDE_long_long,
                            &counter_handle);
    ...
}
```

SDE code for creating a recorder (push mode)

```
void *recorder_handle;

void small_test_init( void ){
    papi_handle_t *handle = papi_sde_init(”TEST");
    papi_sde_create_recorder(handle, "RCRDR",
                             sizeof(double),
                             cmpr_func_ptr,
                             &recorder_handle);
    ...
}
```

SDE code for updating created counters/recorders

```
void *counter_handle;
void *recorder_handle;

void push_test_dowork(void){
    double val;
    long long increment = 3;

    val = perform_useful_work();
    papi_sde_inc_counter(counter_handle, increment);
    papi_sde_record(recorder_handle, sizeof(val), &val);
}
```

Accessing a recorder: data pointer

```
void *recorder_handle;
```

sde:::TEST::RCRDR

```
void small_test_init( void ){
    papi_handle_t *handle = papi_sde_init("TEST");
    papi_sde_create_recorder(handle, "RCRDR",
                             sizeof(double),
                             cmpr_func_ptr,
                             &recorder_handle);
    ...
}
```

Accessing a recorder: element count

```
void *recorder_handle;

void small_test_init(void)
{
    papi_handle_t *handle = papi_sde_init("TEST");
    papi_sde_create_recorder(handle, "RCRDR",
                                     sizeof(double),
                                     cmpr_func_ptr,
                                     &recorder_handle);
    ...
}
```

sde:::TEST::RCRDR

sde:::TEST::RCRDR:CNT

Accessing a recorder: simple statistics

```
void *recorder_handle;
...
void small_test()
{
    papi_handle_t *handle = papi_sde_init("TEST");
    papi_sde_create_recorder(handle, "RCRDR",
                             sizeof(double),
                             comprFuncPtr,
                             &recorder_handle);
    ...
}
```

sde:::TEST::RCRDR

sde:::TEST::RCRDR:CNT

sde:::TEST::RCRDR:MIN

sde:::TEST::RCRDR:Q1

sde:::TEST::RCRDR:MED

sde:::TEST::RCRDR:Q3

sde:::TEST::RCRDR:MAX

Accessing a recorder: simple statistics

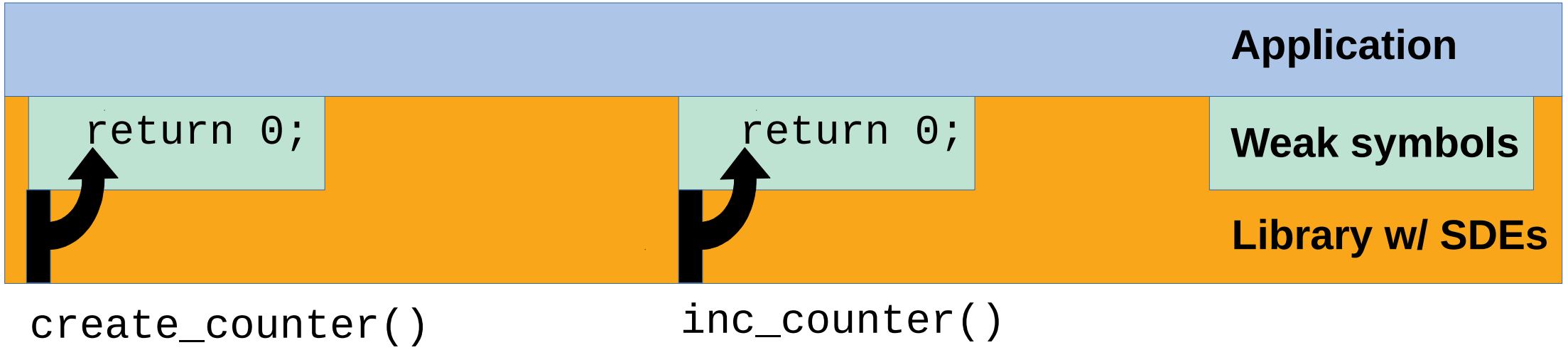
```
void *recorder_handle;

void small_test_init(void)
    papi_handle ...
    papi
    .
}
```

```
sde:::TEST::RCRDR
sde:::TEST
sde
sde:::TEST::RCRDR:Q1
sde:::TEST::RCRDR:MED
sde:::TEST::RCRDR:Q3
sde:::TEST::RCRDR:MAX
```

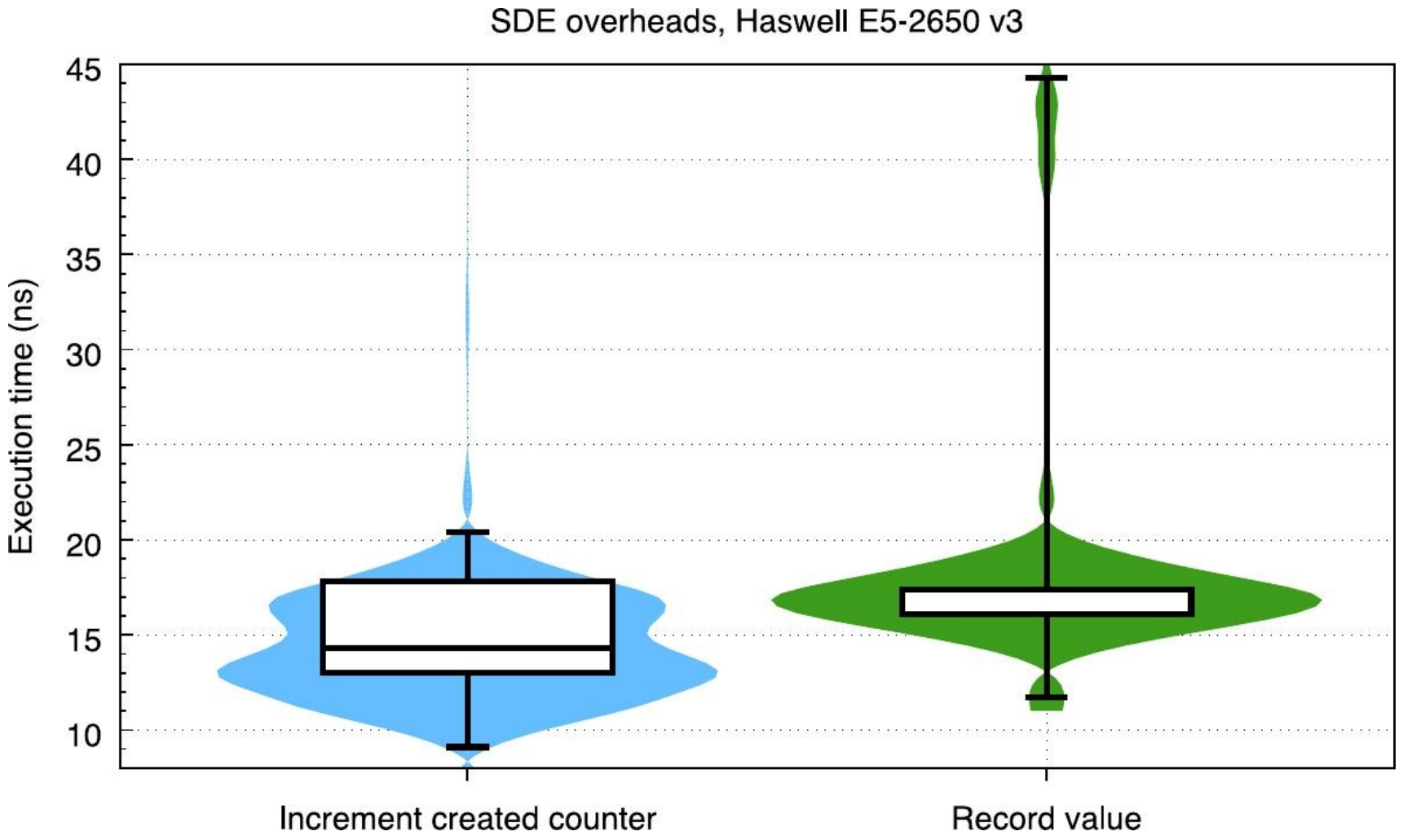
Wouldn’t it be great if PAPI had a C++ interface?

Linking applications without libpapi.so

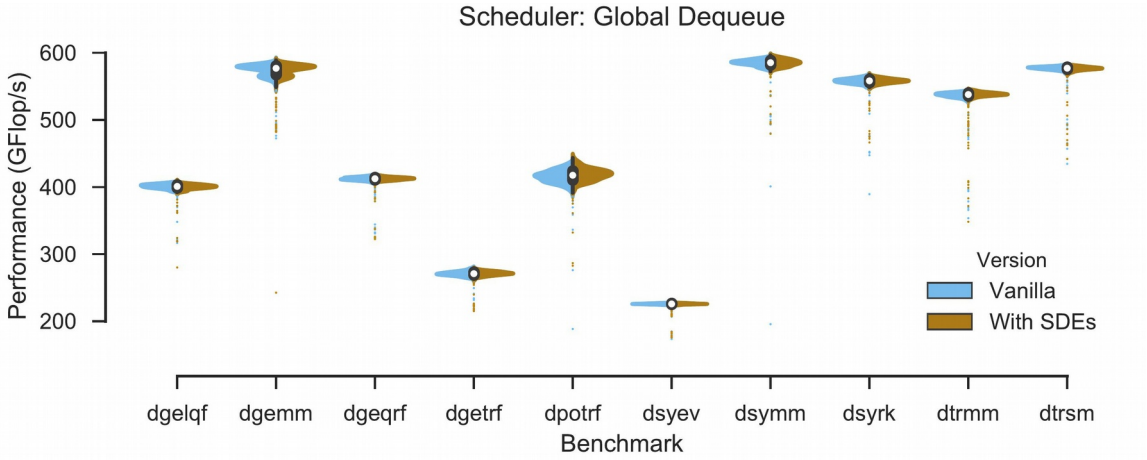
PAPI SDE component comes with a weak symbols header

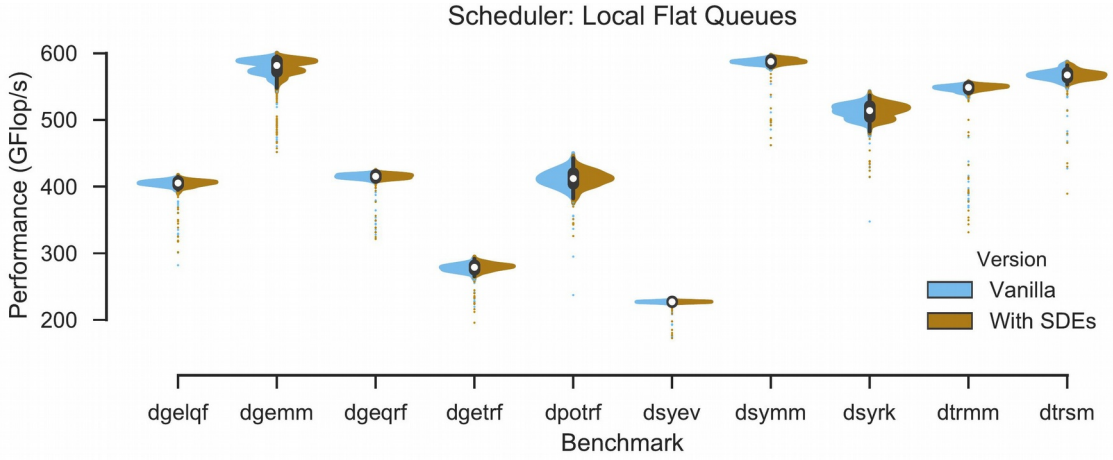


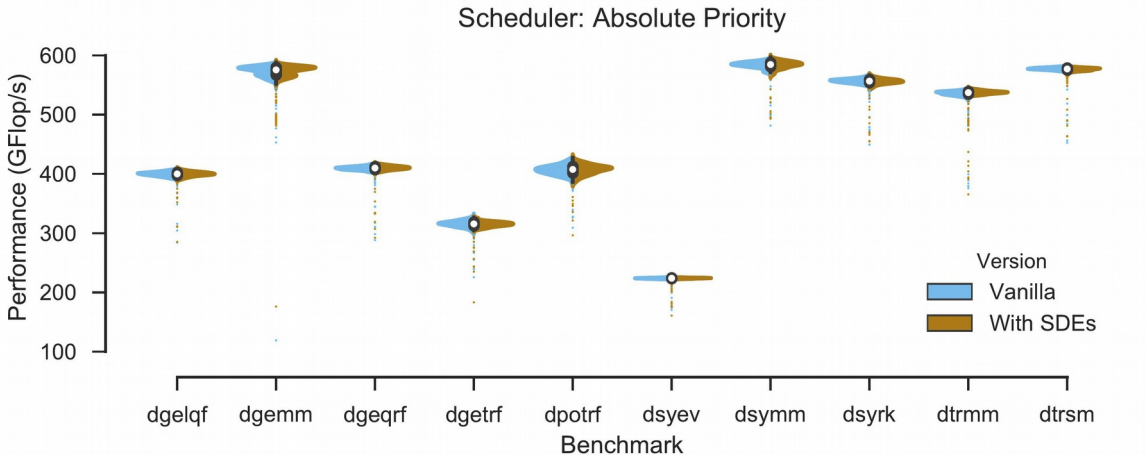
Performance overheads in simple benchmark

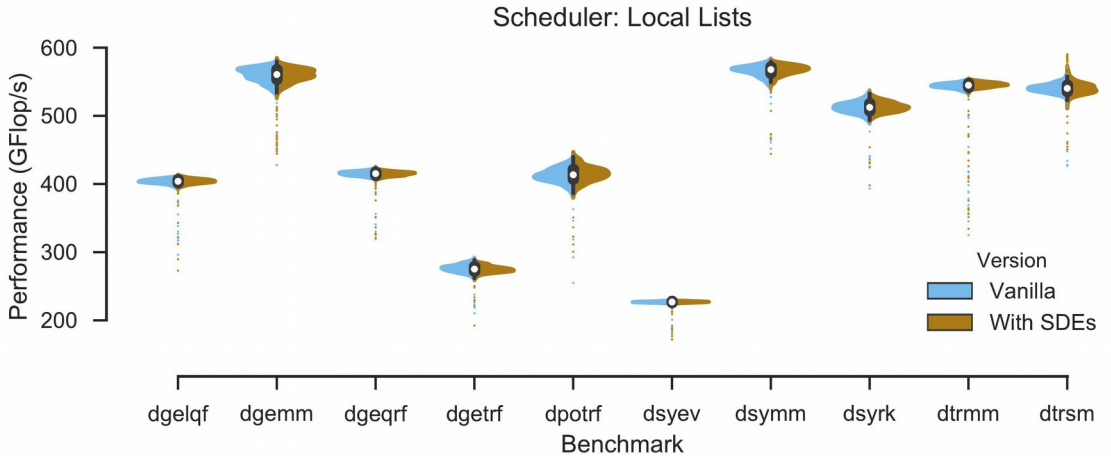


Performance overhead in PaRSEC

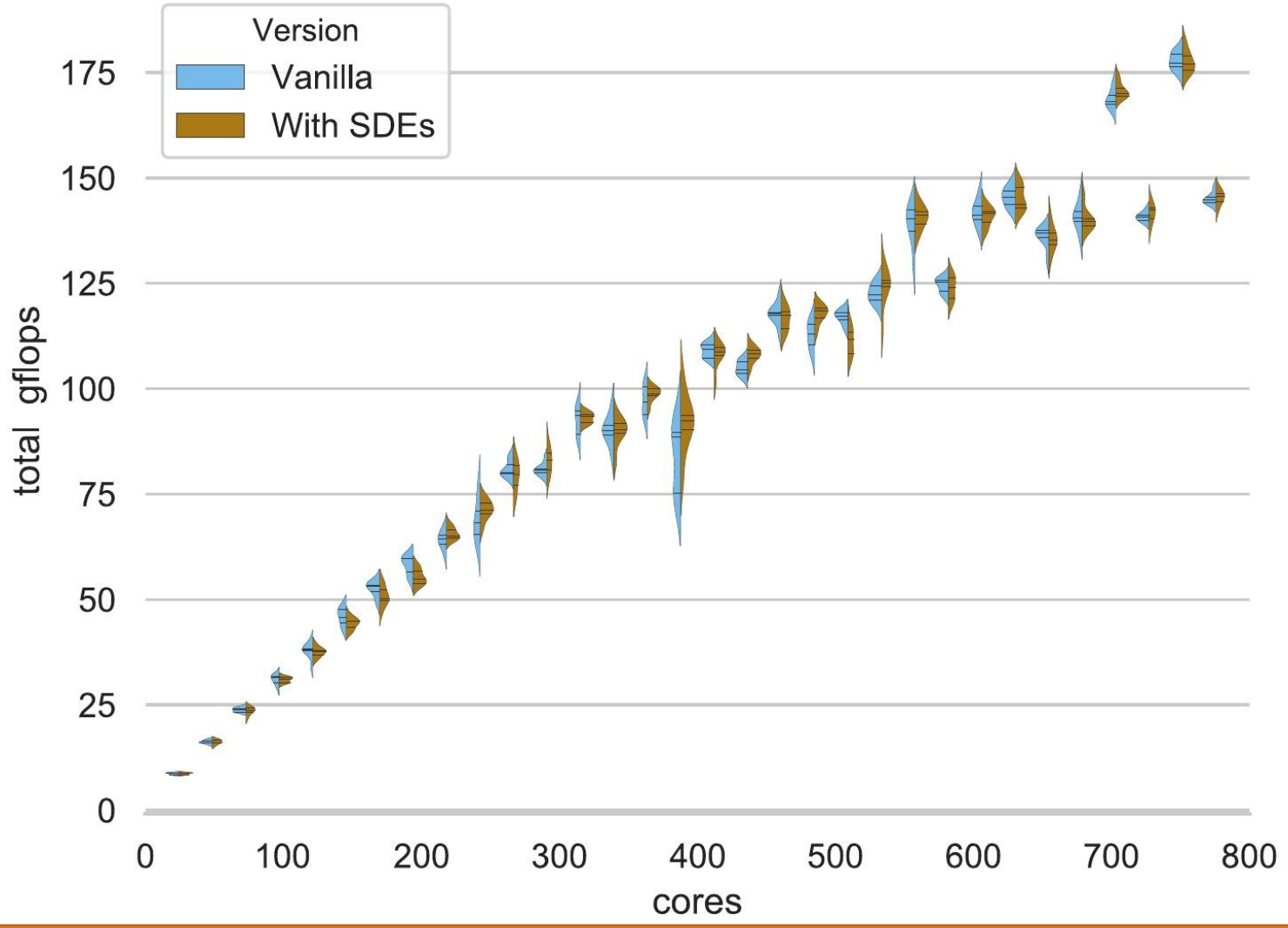




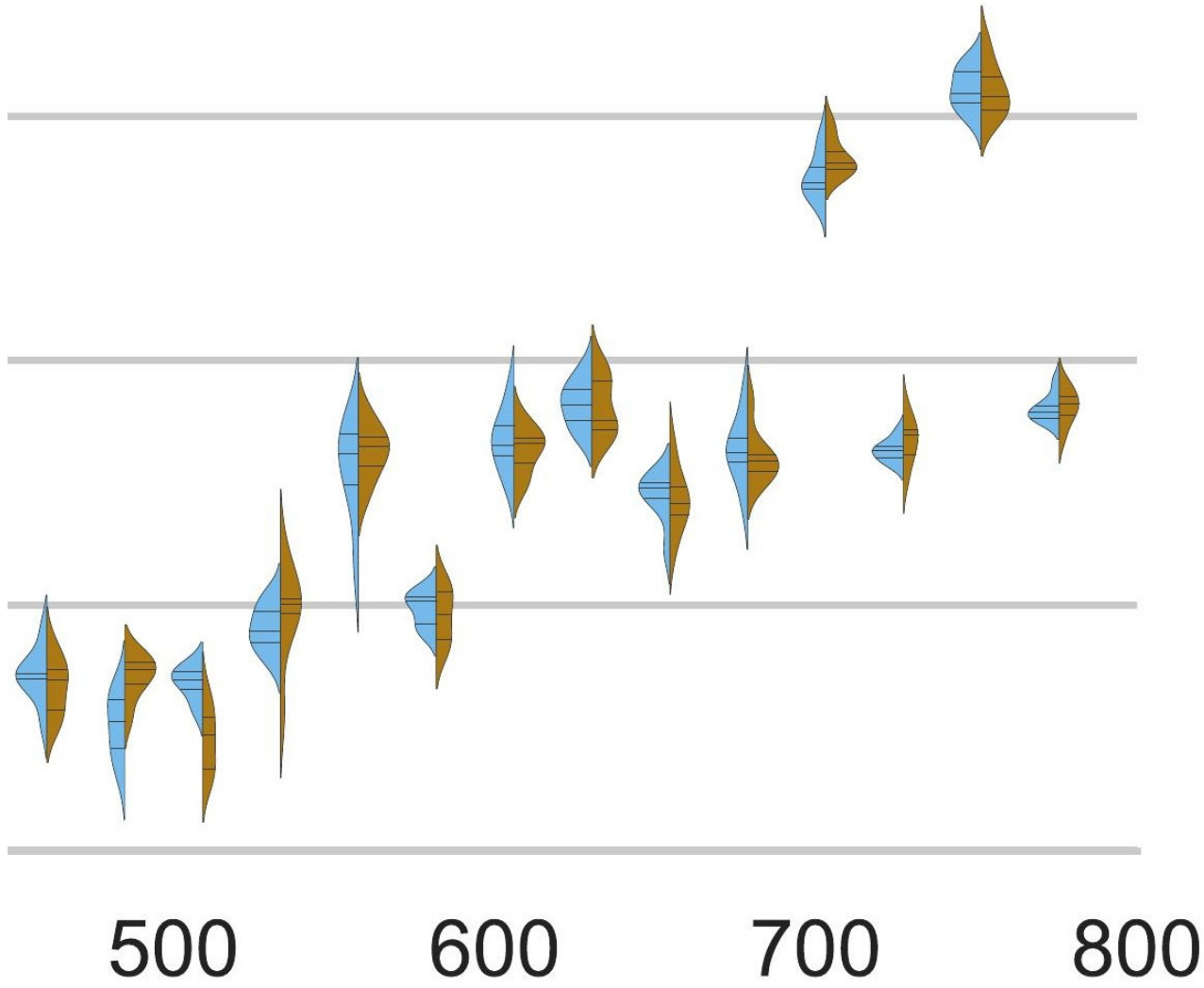




Performance overhead in HPCG



Performance overhead in HPCG (zoom)



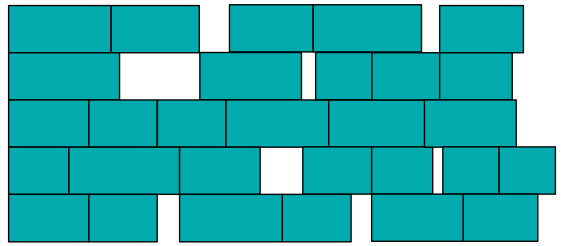
Acquiring insight from SDEs

- Idle time (by measuring hardware counters like stall cycles)
  - Could be due to inevitable memory traffic.
- Idle time (by measuring SDEs from runtime)
  - Great opportunity for "investigative callback"
  - callback must be runtime-aware
- Concurrent idle time
  - Unique insight about application design flaws
  - Additional context can lead to app. redesign

Acquiring insight from SDEs

- Idle time (by measuring hardware counters like stall cycles)
  - Could be due to inevitable memory traffic.

- Idle time (by measuring SDEs from runtime)
  - Great opportunity for “investigative callback”
  - callback must be runtime-aware

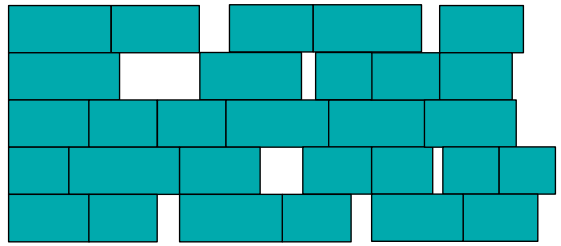


- Concurrent idle time
  - Unique insight about application design flaws
  - Additional context can lead to app. redesign

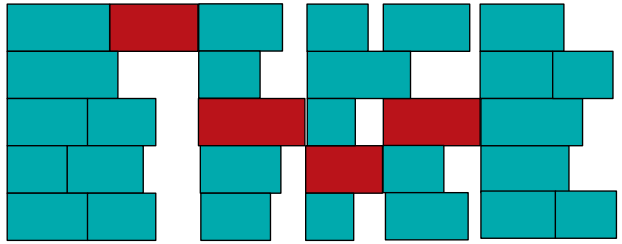
Acquiring insight from SDEs

- Idle time (by measuring hardware counters like stall cycles)
  - Could be due to inevitable memory traffic.

- Idle time (by measuring SDEs from runtime)
  - Great opportunity for “investigative callback”
  - callback must be runtime-aware



- Concurrent idle time
  - Unique insight about application design flaws
  - Additional context can lead to app. redesign



Open Problem for our Community:

How do we associate useful context information with SDEs?

What information to associate with CONCR_IDLE, or TASKS_STOLEN?

- ~~Code location~~
- Hardware events (e.g. cache misses)
- List of all threads' activity
- Patterns in history (e.g. last task before stealing event)
- Patterns in call-path/stack/originating thread

Conclusions

- Libraries/runtimes generate multiple useful software “events”.
- PAPI SDE allows any software layer to export events.
- SDEs can be read using the standard PAPI functionality.
- SDEs have minimal to **zero** performance overhead.
- SDEs call for new types of analysis by tools.
- PAPI++ soon at a repo near you.