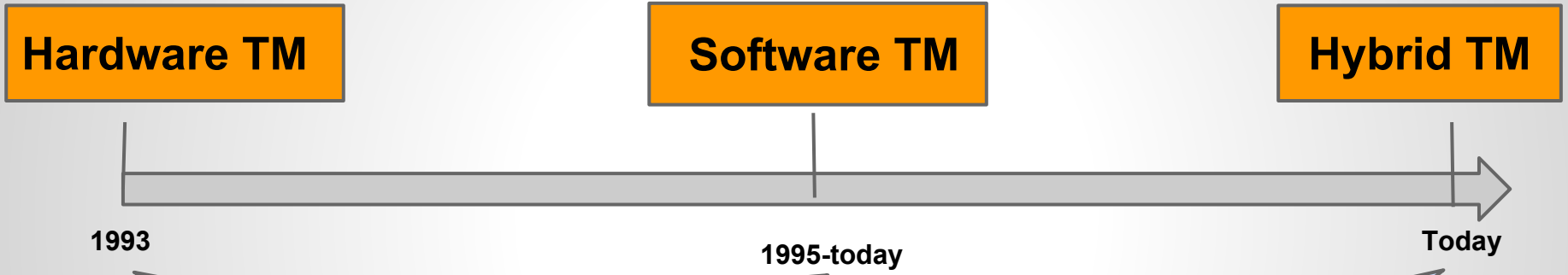
Cost of Concurrency in Hybrid Transactional Memory

Trevor Brown (University of Toronto)

Srivatsan Ravi (Purdue University)

Transactional Memory: a history



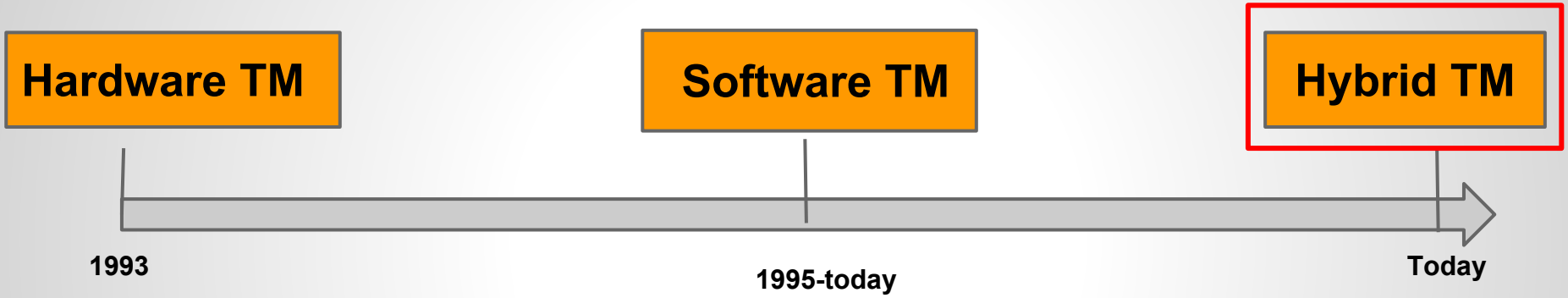
➔ Software implementation by Shavit-Touit… …static”

➔ HTM support: Intel Haswell, IBM Power8,..
- ◆ Different memory models and supported instructions

➔ Hardware limitations and “spurious” aborts

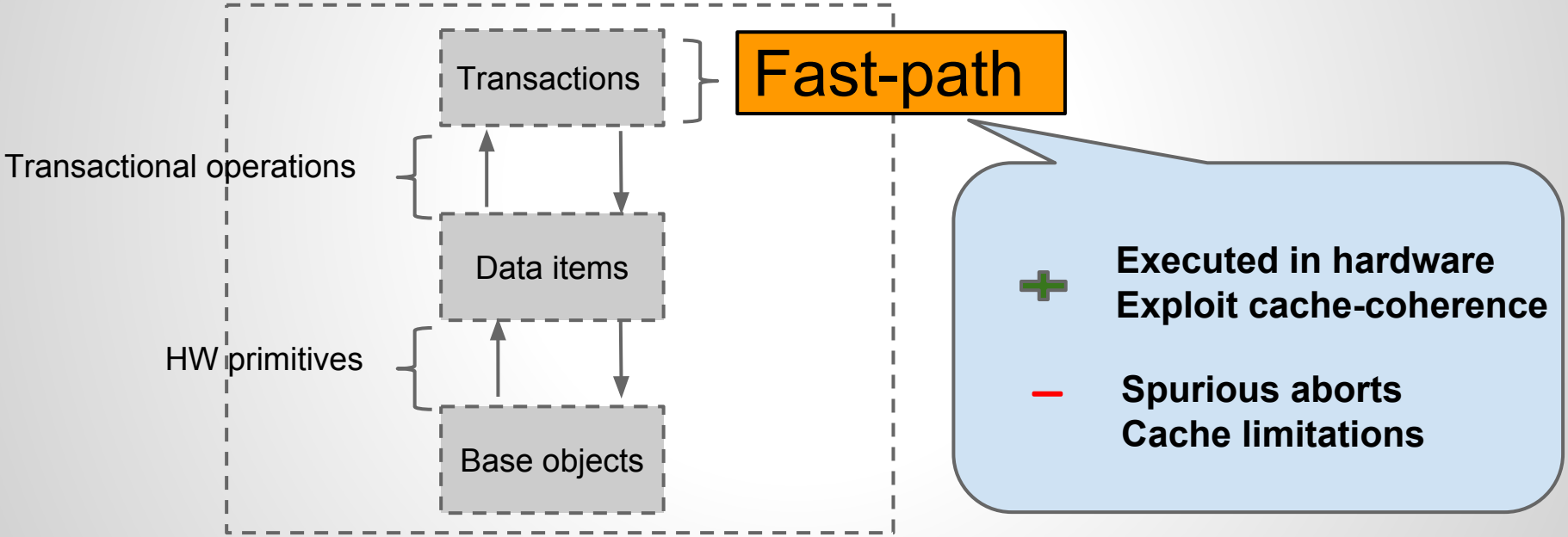
➔ Fallback to software transactions

Transactional Memory: a history

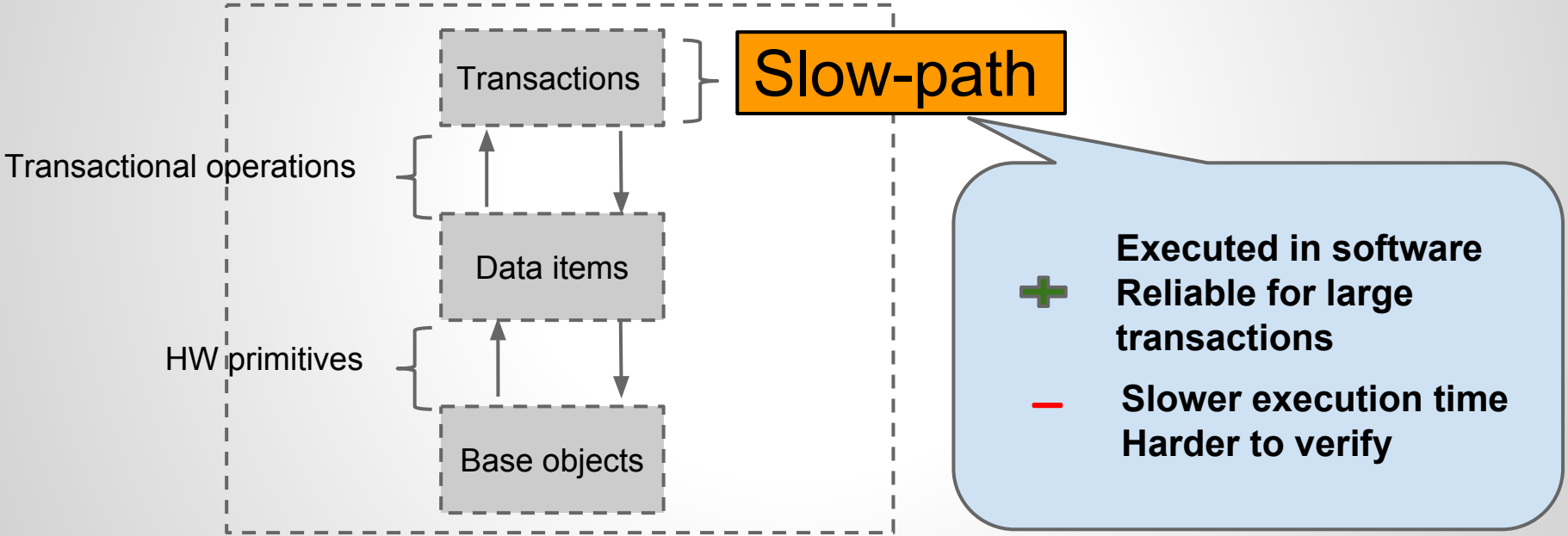


Cost of Concurrency in Hybrid TMs?

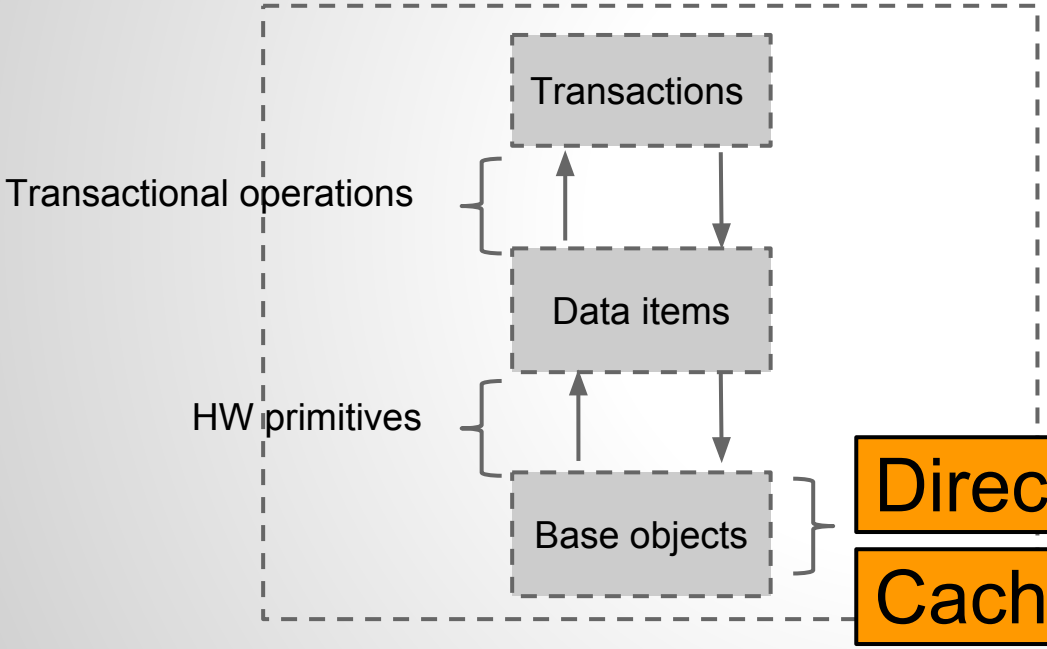
Hybrid Transactional Memory (HyTM) Model



Hybrid Transactional Memory (HyTM) Model



Hybrid Transactional Memory (HyTM) Model



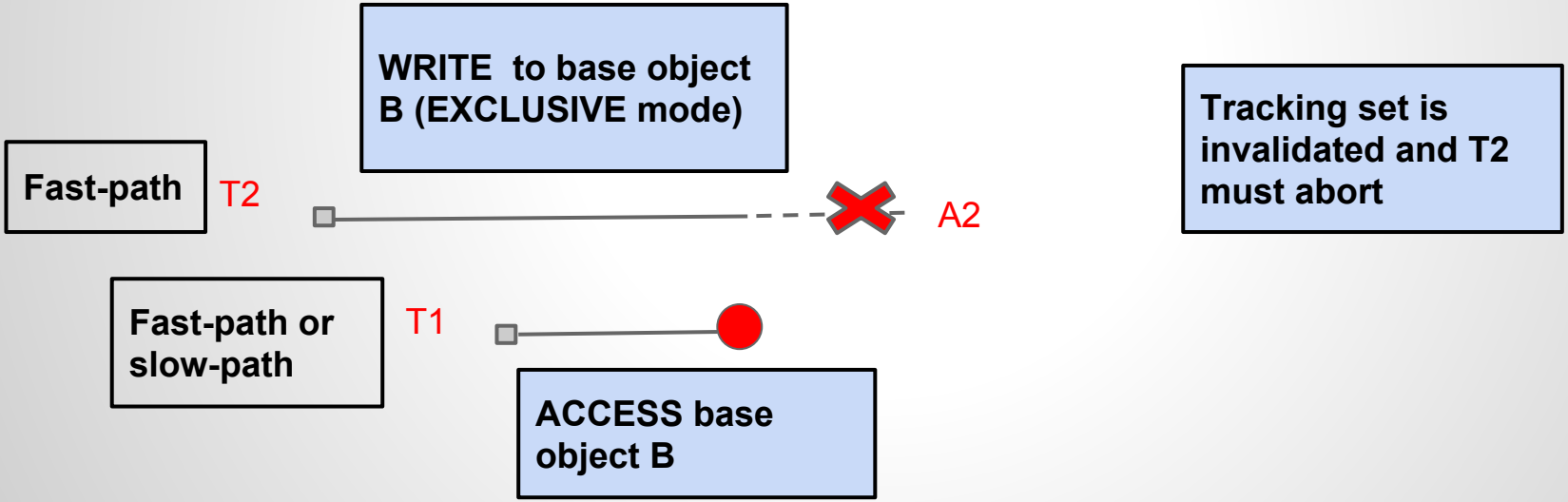
- For fast-path transactions
- Operate on “cached” memory state
  - Direct on Power8
- Maintain TRACKING SET
  - Shared/exclusive mode
  - Capacity limit

Direct access

Cached access

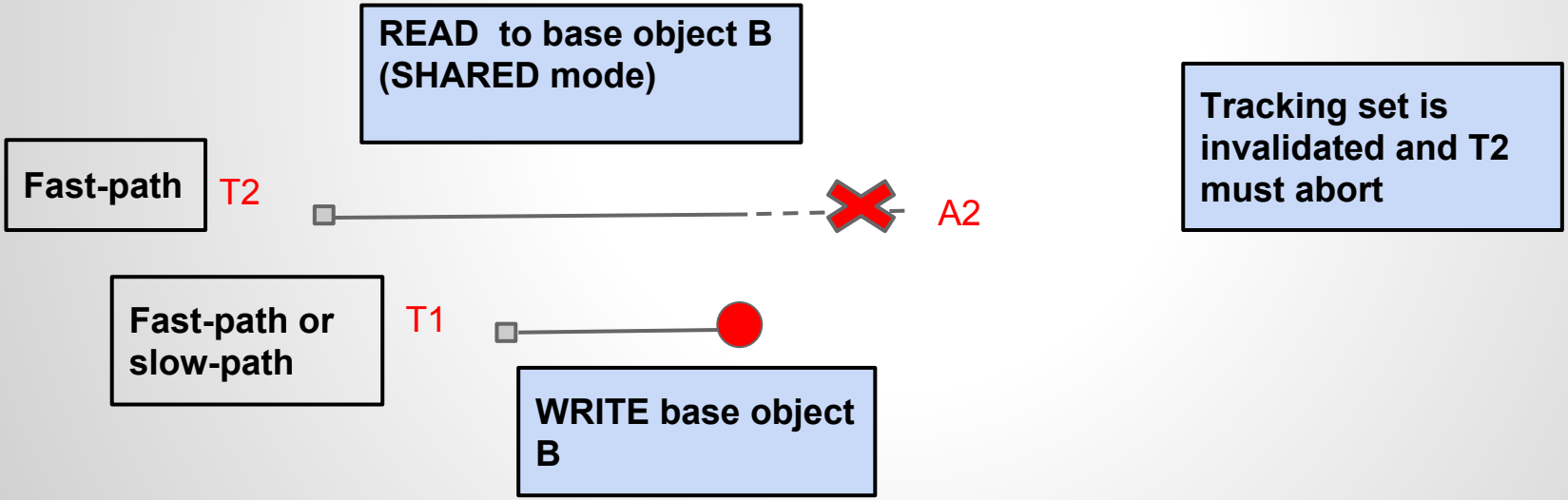
Hybrid Transactional Memory (HyTM) Model

Tracking set aborts in fast-path transactions
Automatic contention detection for cached accesses



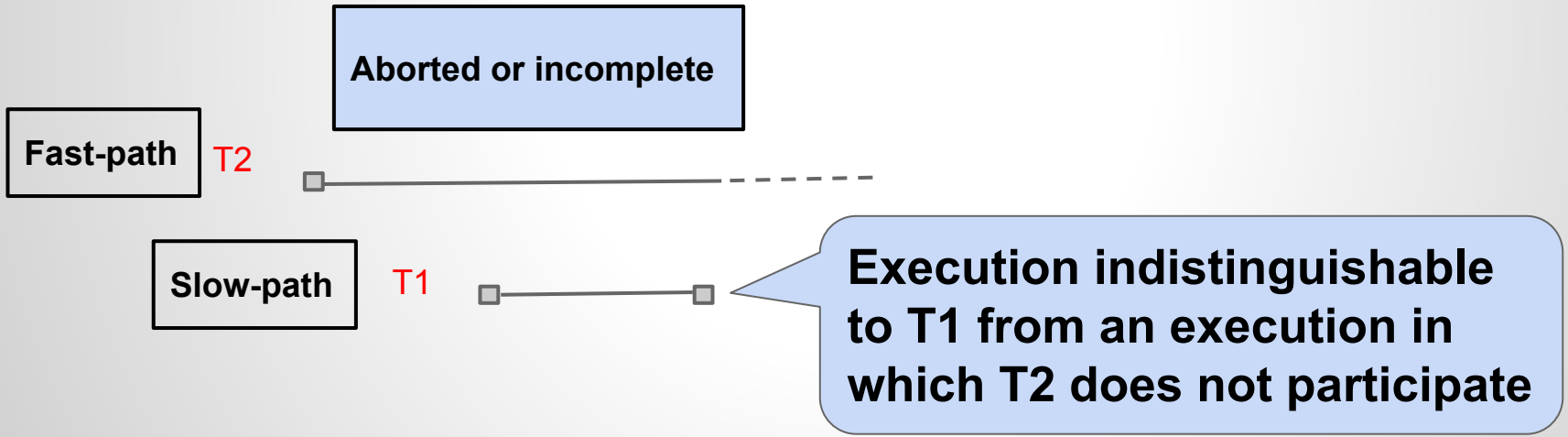
Hybrid Transactional Memory (HyTM) Model

Tracking set aborts in fast-path transactions
Automatic contention detection for cached accesses



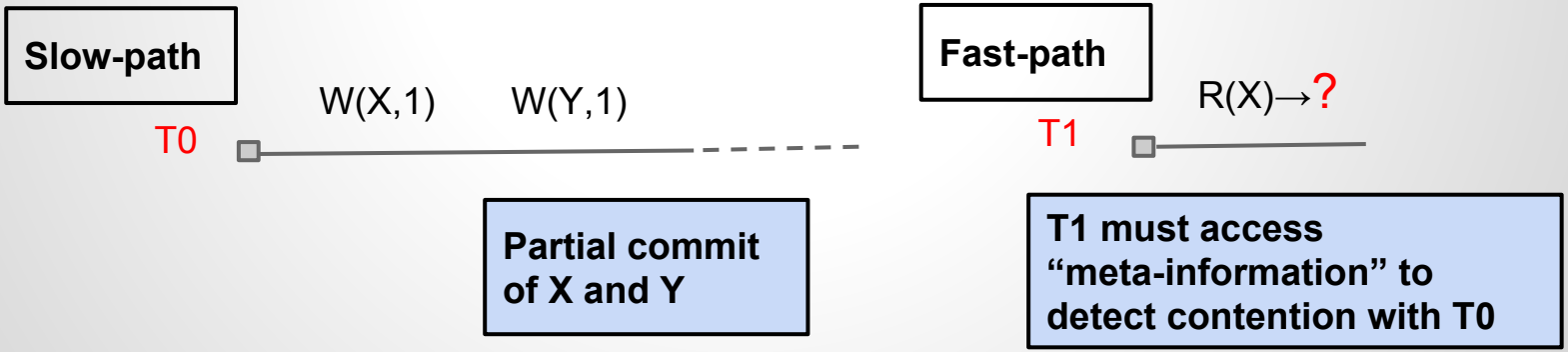
Hybrid Transactional Memory (HyTM) Model

Committed fast-path transactions (only cached accesses) appear to execute atomically (single step)

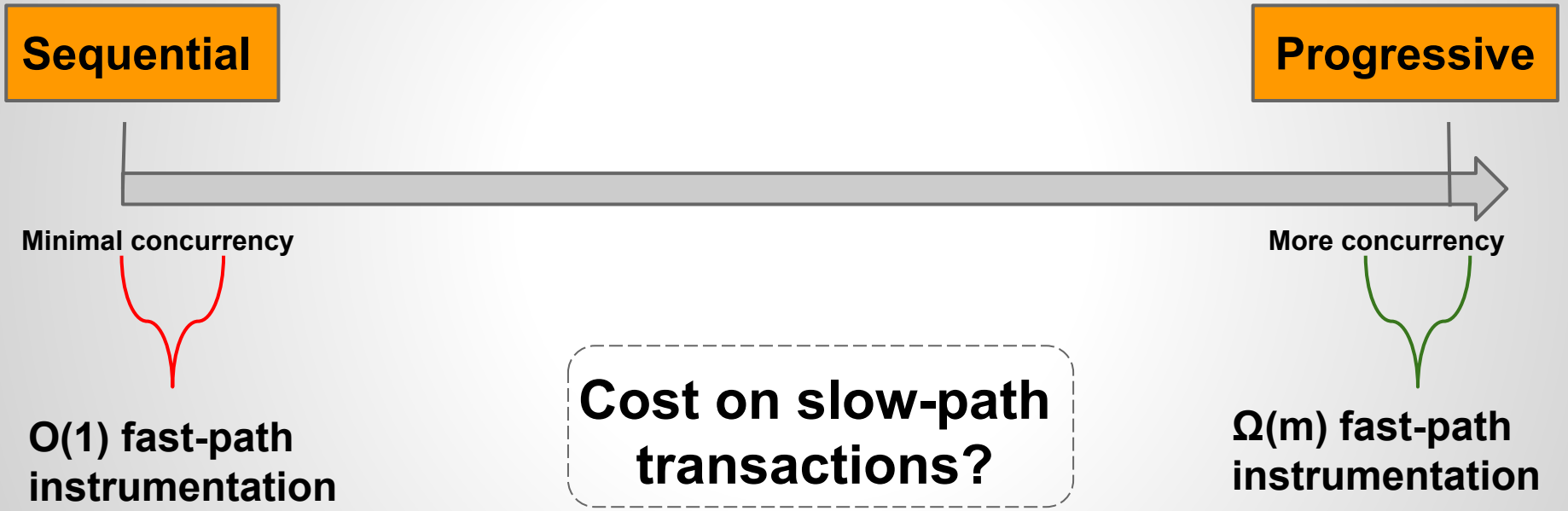


Instrumentation

- Fast-path transactions intuitively need code "instrumentation"
  - Detect overlap contention with slow-path
    - Global timestamp, ownership records, etc.
  - Instrumentation affects performance

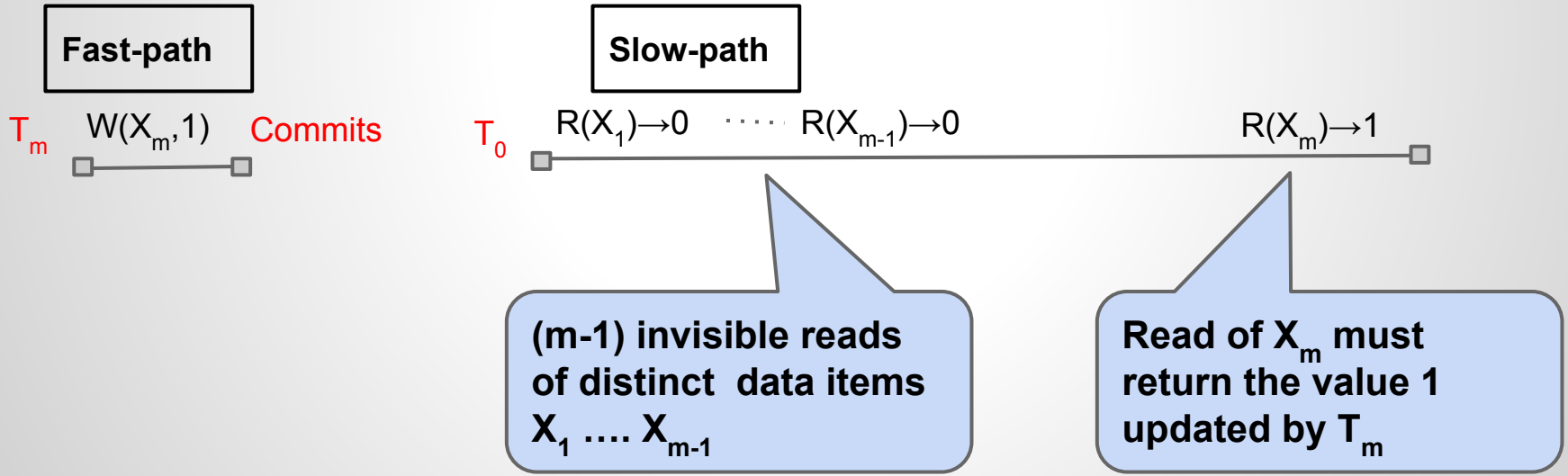


Cost of concurrency in HyTM



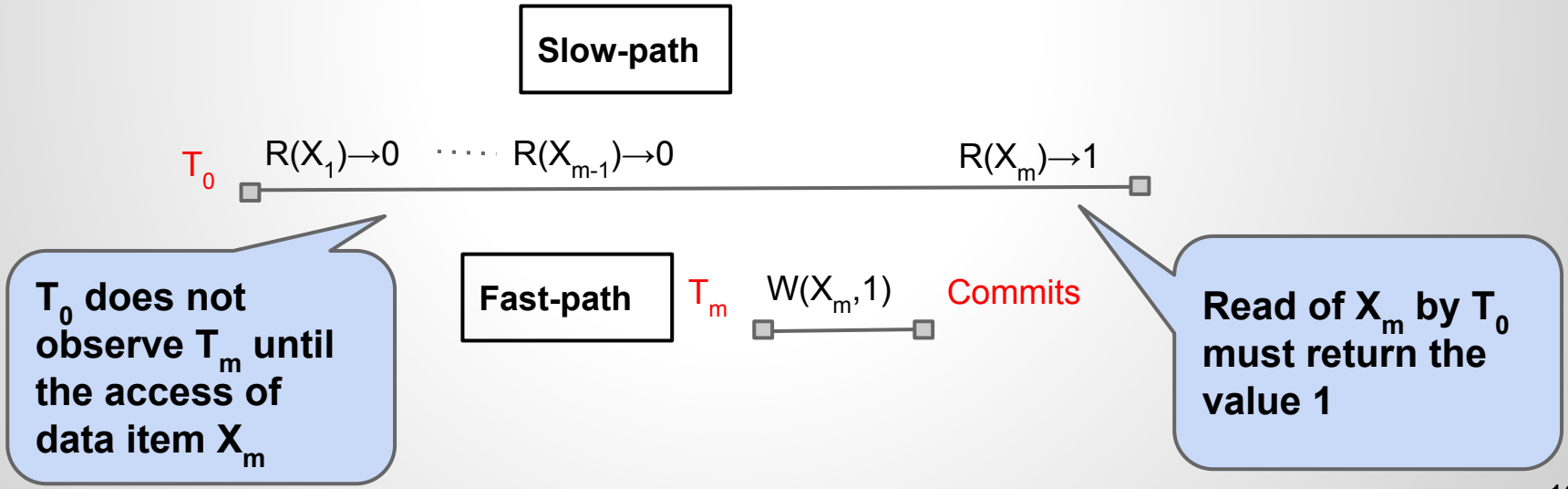
Linear validation cost in HyTM

Progressive opaque HyTM+Invisible reads ⇒ Linear time and space complexity for slow-path transactions



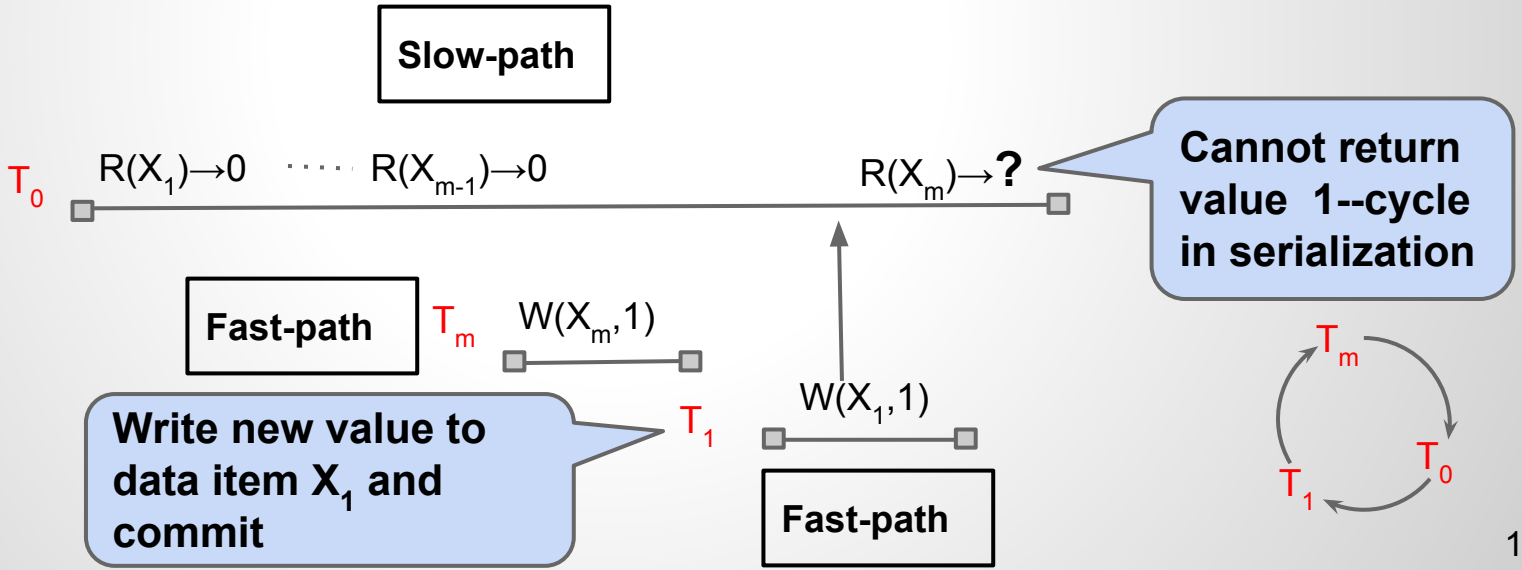
Linear validation cost in HyTM

Progressive opaque HyTM+Invisible reads ⇒ Linear time and space complexity for slow-path transactions



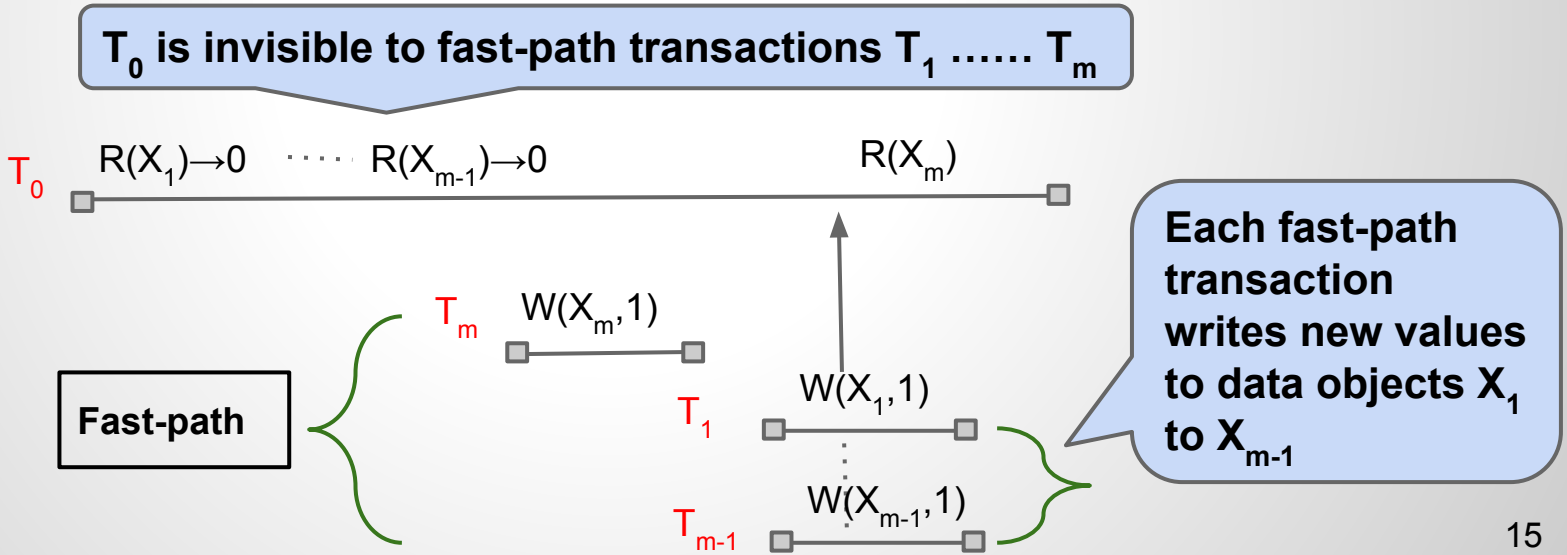
Linear validation cost in HyTM

Progressive opaque HyTM+Invisible reads ⇒ Linear time and space complexity for slow-path transactions



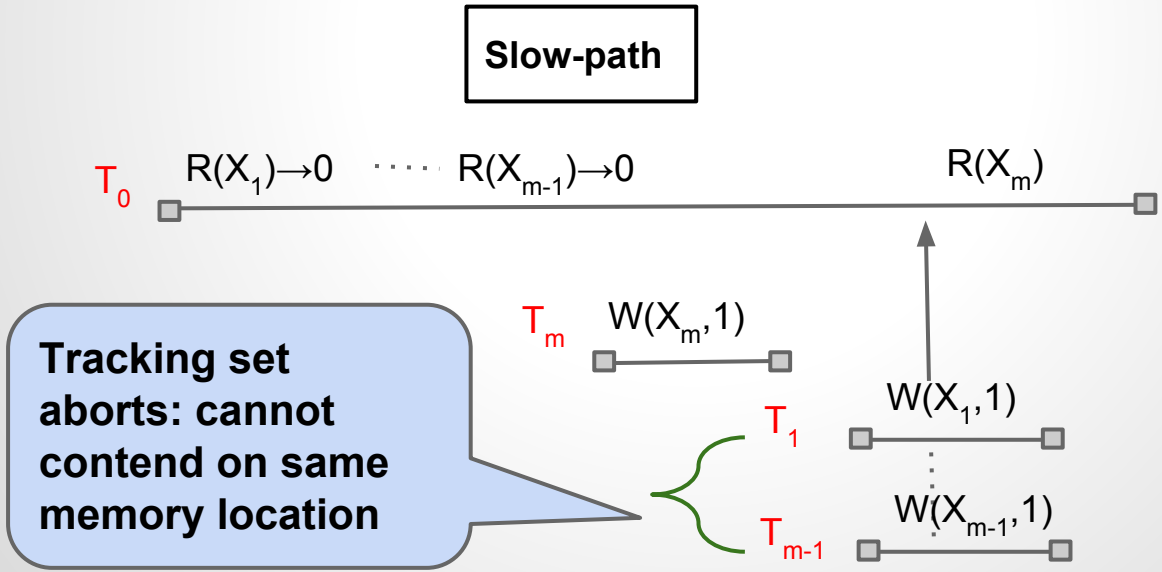
Linear validation cost in HyTM

Progressive opaque HyTM+Invisible reads ⇒ Linear time and space complexity for slow-path transactions



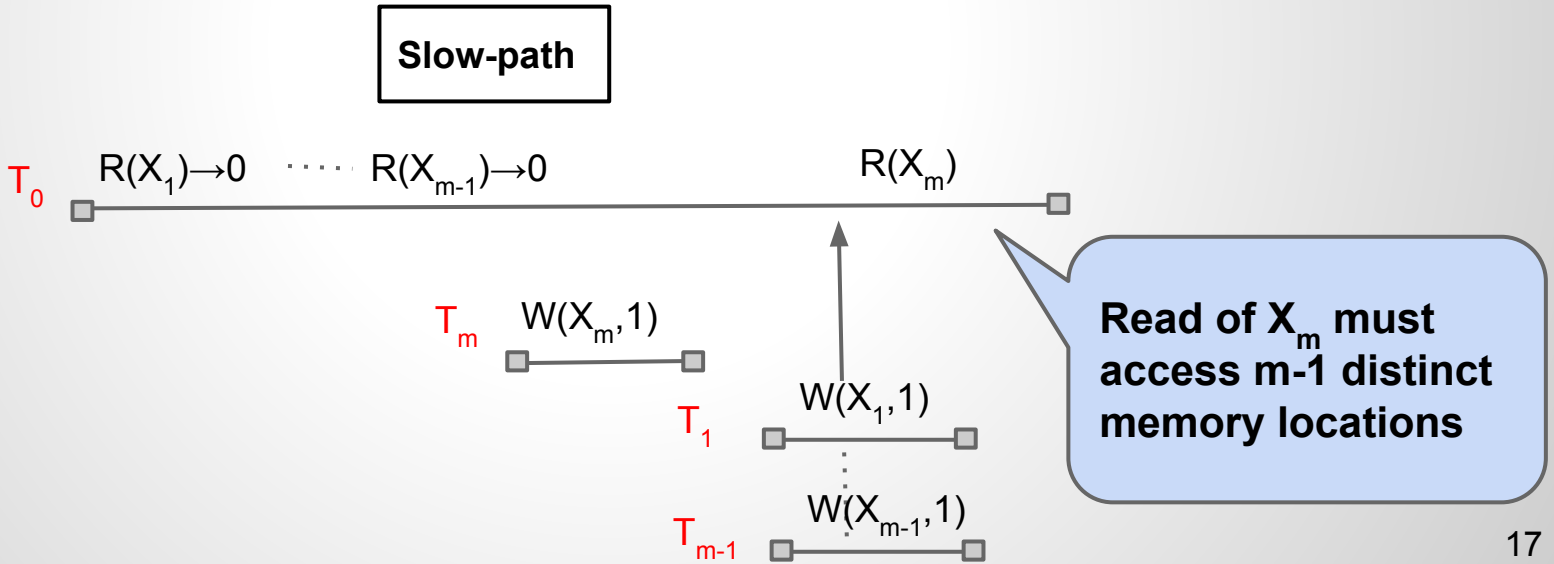
Linear validation cost in HyTM

Progressive opaque HyTM+Invisible reads ⇒ Linear time and space complexity for slow-path transactions



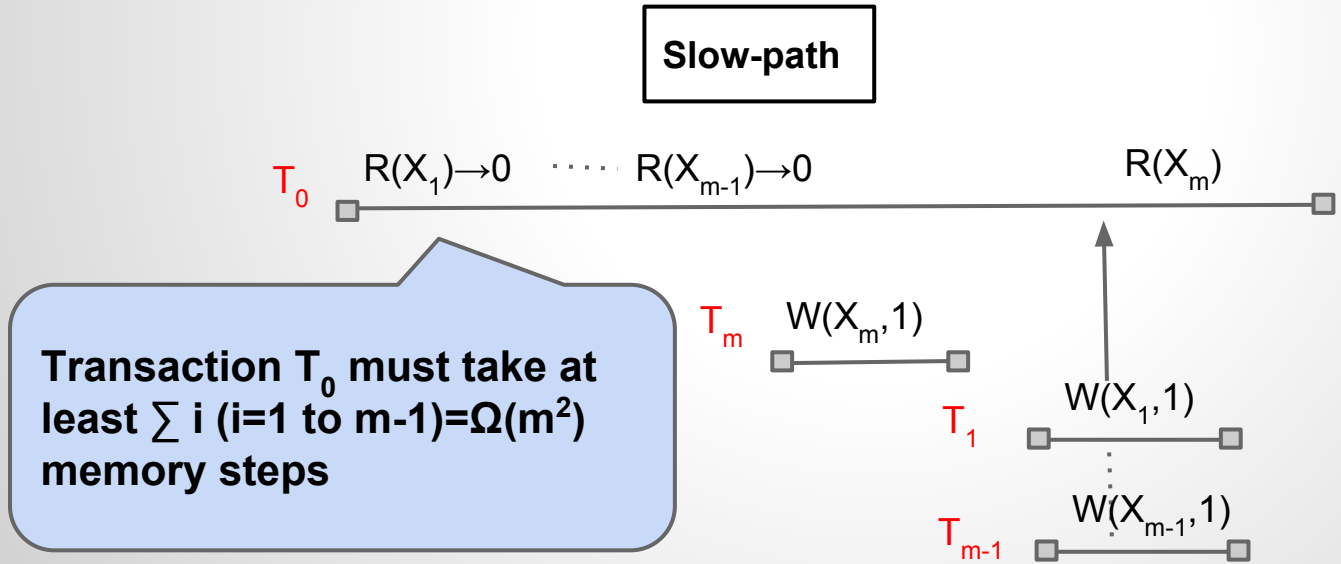
Validation cost in HyTM

Progressive opaque HyTM+Invisible reads ⇒ Linear time and space complexity for slow-path transactions

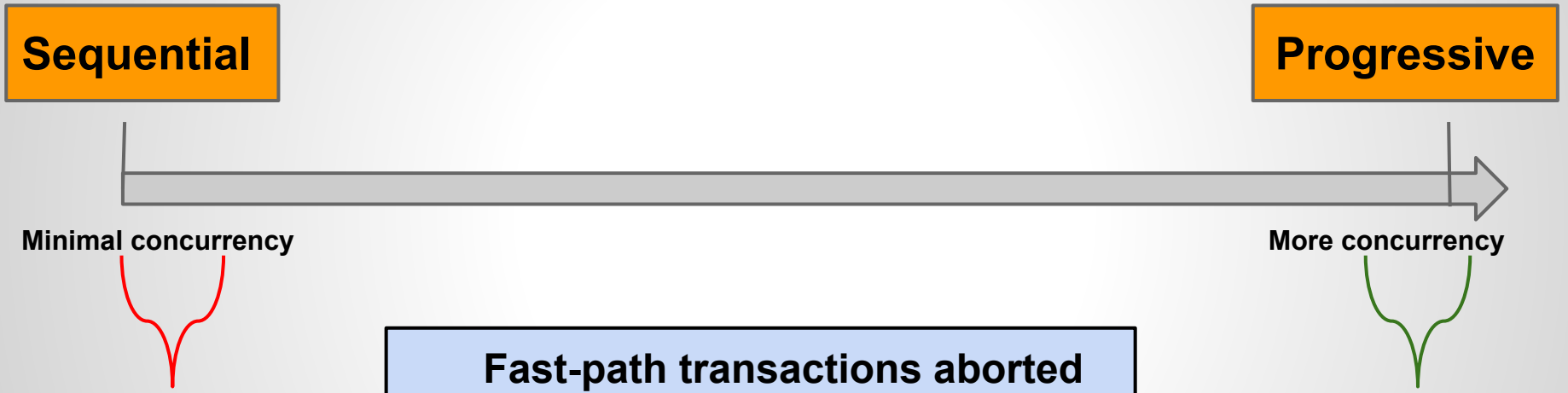


Validation cost in HyTM

Progressive opaque HyTM+Invisible reads ⇒ Linear time and space complexity for slow-path transactions



Cost of concurrency in HyTM

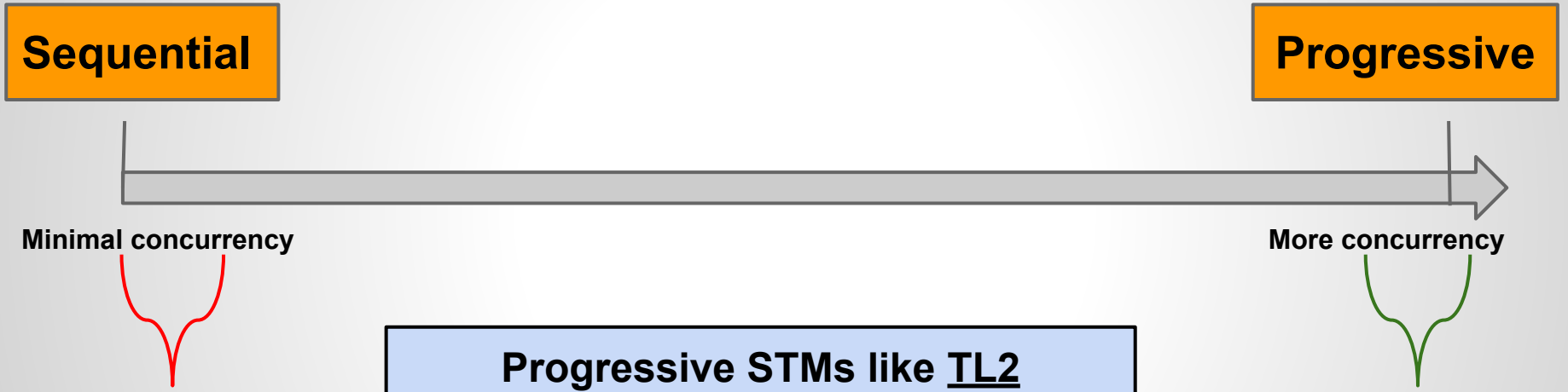


O(1) fast-path instrumentation+ O(1) slow-path reads

Fast-path transactions aborted by non-conflicting ones or linear fast-path instrumentation cost and linear slow-path steps

Ω(m) fast-path instrumentation + Ω(m) slow-path steps per-read

Cost of concurrency in HyTM



O(1) fast-path instrumentation+ O(1) slow-path reads

Progressive STMs like TL2 circumvent the cost of validation for better performance, but impossible in progressive HyTMs!

Ω(m) fast-path instrumentation + Ω(m) slow-path steps per-read

Cost of Concurrency in HyTM

	Algorithm 1	Algorithm 2	Transactional Lock Elision	Hybrid Norec
Instrumentation in fast-path reads	per-read	constant	constant	constant
Instrumentation in fast-path writes	per-write	per-write	constant	constant
Validation in slow-path reads	Ω(|Rset|)	Ω(|Rset|)	none	Ω(|Rset|) only if concurrency
h/w-s/w concurrency	prog	prog for slow-path readers	zero	Not prog; small contention window
Direct accesses inside fast-path?	yes	no	no	yes

Experimental setup

- **Large-scale 2-socket Intel E7-4830 v3 with 12 cores per socket and 2 hyperthreads (HTs) per core, for a total of 48 threads**

- **Each core has a private 32KB L1 cache and 256KB L2 cache (which is shared between HTs on a core)**
  - **All cores on a socket share a 30MB L3 cache**
  - **Non-uniform memory architecture (NUMA)**
  - **128GB of RAM, and runs Ubuntu 14.04 LTS.**

- **All code was compiled with the GNU C++ compiler (G++) 4.8.4 with build target x86_64-linux-gnu and compilation options *-std=c++0x -O3 -mx32***

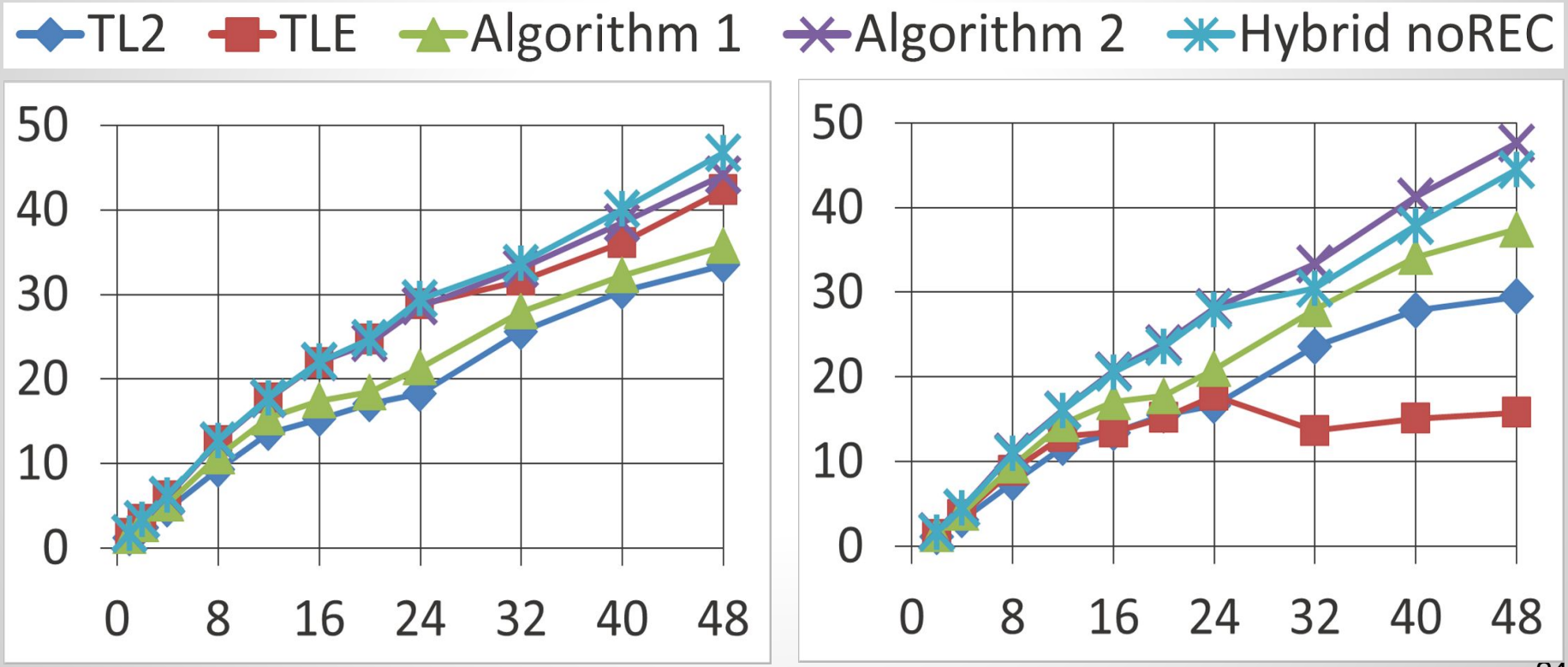
Experimental methodology

- **Six timed trials for several thread counts n**
  - ***Prefilling*: n concurrent threads perform 50% Insert and 50% Delete operations on keys drawn uniformly randomly from [0, 10^5) until the size of the tree converges to a steady state (containing approximately 10^5 /2 keys)**
  - ***Measuring*: Each thread performs (U/2)% Insert, (U/2)% Delete and (100 − U)% Search operations, on keys/values drawn uniformly from [0, 10^5); U=0,10,40**

- **Plots for Binary Search Tree (BST) microbenchmark**
  - **With (without) one (any) thread performing RangeIncrement operations**
    - **Capacity aborts on fast-path**

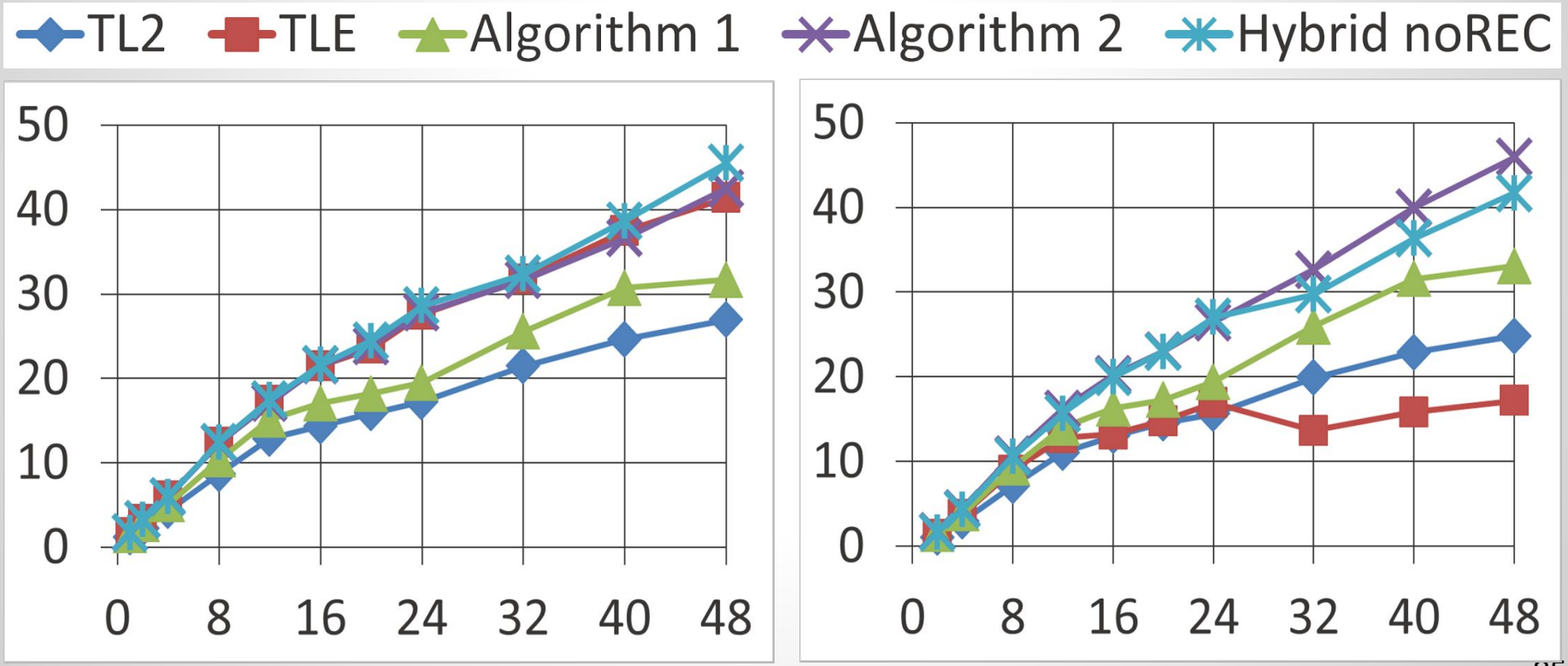
Cost of Concurrency in HyTM

0% updates: #threads vs. ops/microsec



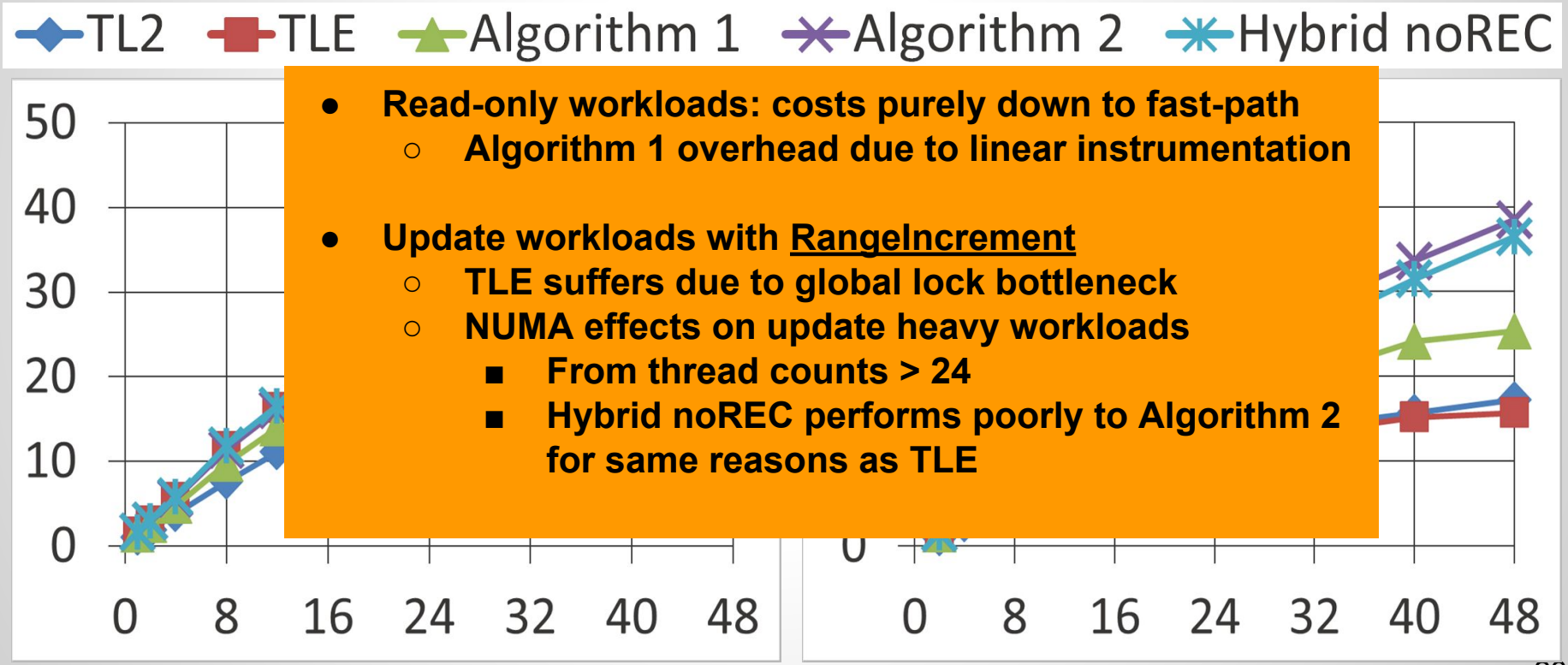
Cost of Concurrency in HyTM

10% updates: #threads vs. ops/microsec



Cost of Concurrency in HyTM

40% updates: #threads vs. ops/microsec



- **Read-only workloads: costs purely down to fast-path**
  - **Algorithm 1 overhead due to linear instrumentation**
- **Update workloads with <u>RangeIncrement</u>**
  - **TLE suffers due to global lock bottleneck**
  - **NUMA effects on update heavy workloads**
    - **From thread counts > 24**
    - **Hybrid noREC performs poorly to Algorithm 2 for same reasons as TLE**

Circumventing the impossibilities?

- Middle-path approach? Ongoing work
  - Almost uninstrumented “fast” fast-path
    - No concurrency with slow-path
    - Concurrent with middle-path
- Ongoing experiments on Intel Haswell and IBM Power8 (STAMP and data structure microbenchmarks)
  - Completely different memory models
    - Power8 allows “direct” accesses inside hardware

Transactional memory is here to stay?

- HyTM: an efficient “universal construction”?
  - Start with a “base” HyTM with minimal instrumentation overhead, maximal concurrency and little global metadata bottleneck
  - Dynamic implementation choices depending on workloads
    - Multi-path approach
  - Formal methods and verification techniques
  - Impact of cache hierarchy, cache-size and memory model on HyTM performance