STAMP: Stanford Transactional Applications for Multi-Processing

Chí Cao Minh, JaeWoong Chung, Christos Kozyrakis, Kunle Olukotun

http://stamp.stanford.edu

15 September 2008

Motivation

- Multi-core chips are here
  - But writing parallel SW is hard
- Transactional Memory (TM) is a promising solution
  - Large atomic blocks simplify synchronization
  - Speed of fine-grain locks with simplicity of coarse-grain locks
  - But where are the benchmarks?
- STAMP: A new benchmark suite for TM
  - 8 applications specifically for evaluating TM
  - Comprehensive *breadth* and *depth* analysis
  - *Portable* to many kinds of TMs (HW, SW, hybrid)
  - Publicly available: `http://stamp.stanford.edu`

Outline

- Introduction
- Transactional Memory Primer
- Design of STAMP
- Evaluation of STAMP
- Conclusions

Programming Multi-cores

- Commonly achieved via:
  - Threads for parallelism
  - Locks for synchronization
- Unfortunately, synchronization with locks is hard
  - Option 1: Coarse-grain locks
    - Simplicity ☺
    - Decreased concurrency ☹
  - Option 2: Fine-grain locks
    - Better performance ☺ (maybe)
    - Increased complexity ☹ (bugs)
      - Deadlock, priority inversion, convoying, …

Transactional Memory (TM)

- What is a transaction?
  - Group of instructions in computer program:

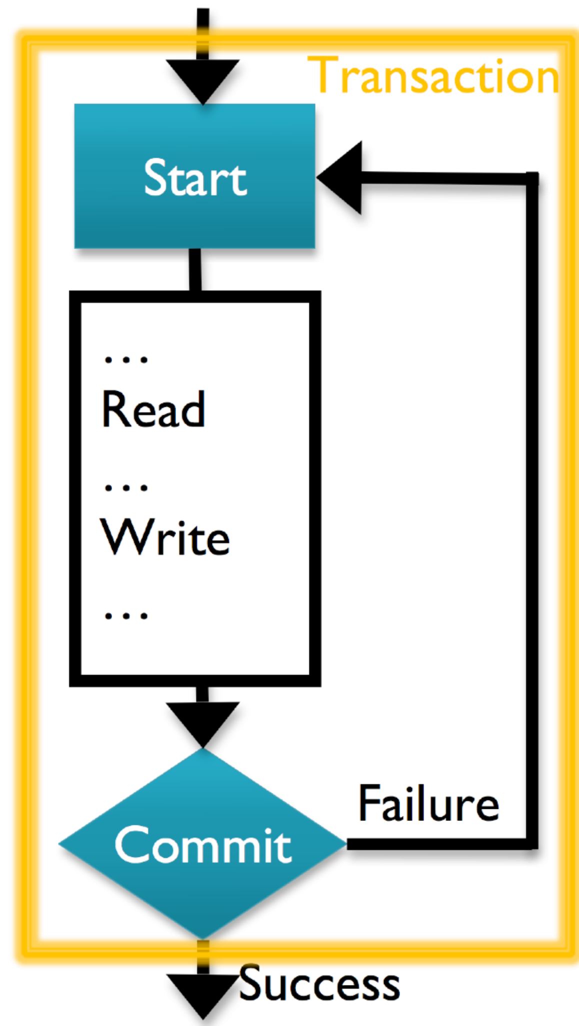
```
atomic {
  if (x != NULL) x.foo();
  y = true;
}
```

  - Required properties: Atomicity, Isolation, Serializability

- Key idea: Use transactions to ease parallel programming
  - Locks → programmers define & implement synchronization
  - TM → programmers declares & system implements
    - Simple like coarse-grain locks & fast like fine-grain locks

Optimistic Concurrency Control

- Each core optimistically executes a transaction
- Life cycle of a transaction:
  - Start
  - Speculative execution (optimistic)
  - Build read-set and write-set
  - Commit
    - Fine-grain R-W & W-W conflict detection
  - Abort & rollback



Parallel Programming With TM

Thread 1: insert 2

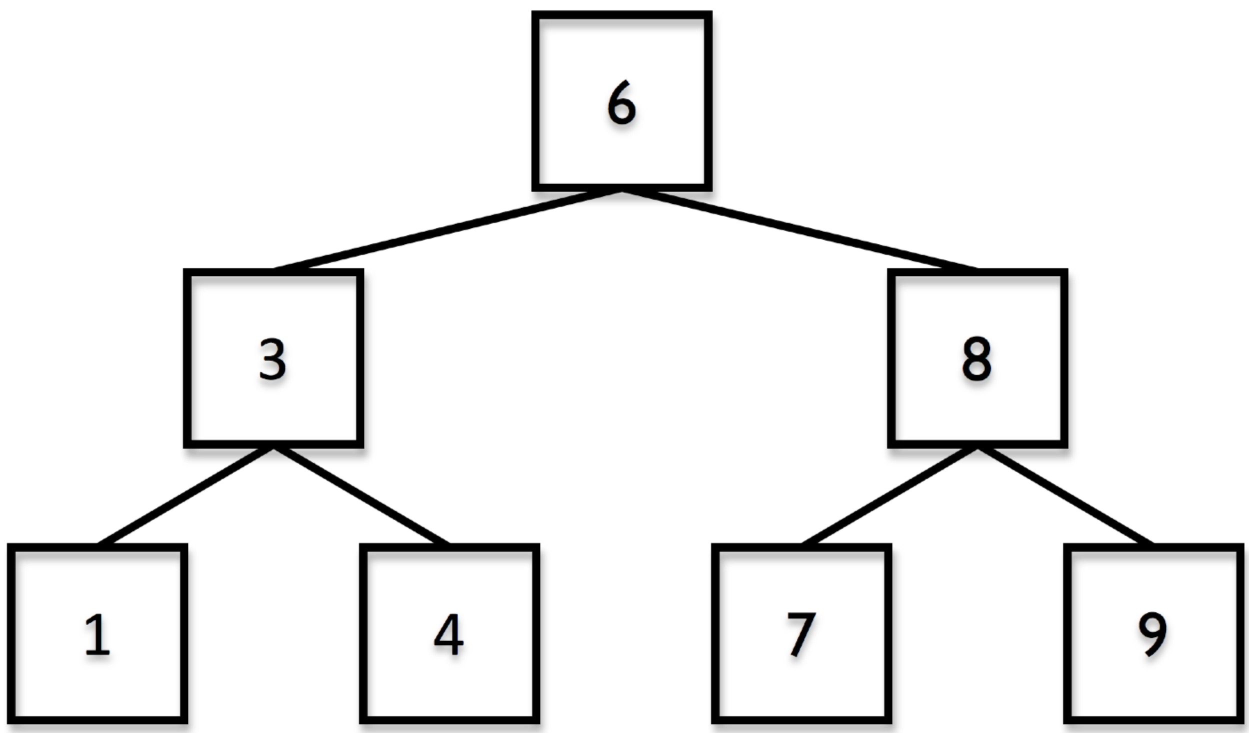
Read-set:

Write-set:

Thread 2: insert 5

Read-set:

Write-set:



Parallel Programming With TM

Thread 1: insert 2

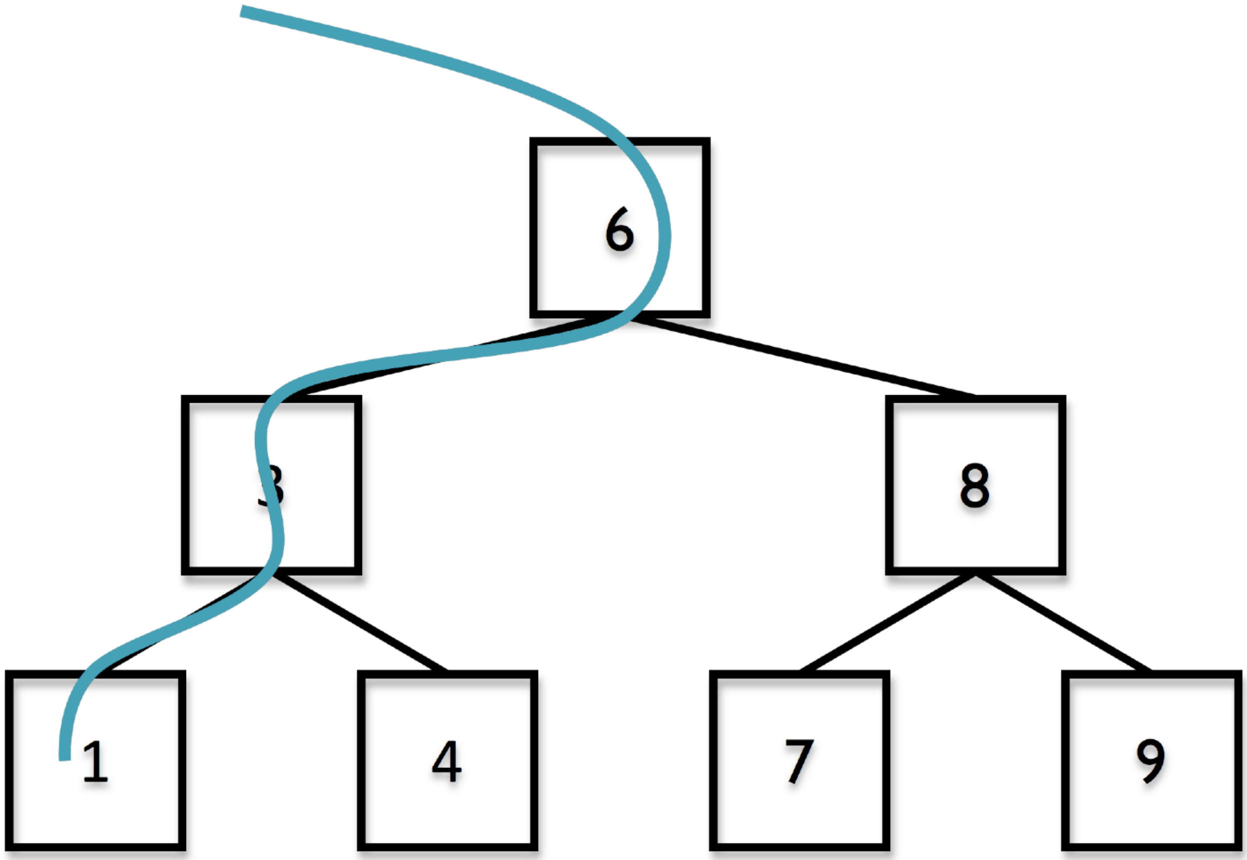
Read-set:

Write-set:

Thread 2: insert 5

Read-set:

Write-set:



Parallel Programming With TM

Thread 1: insert 2

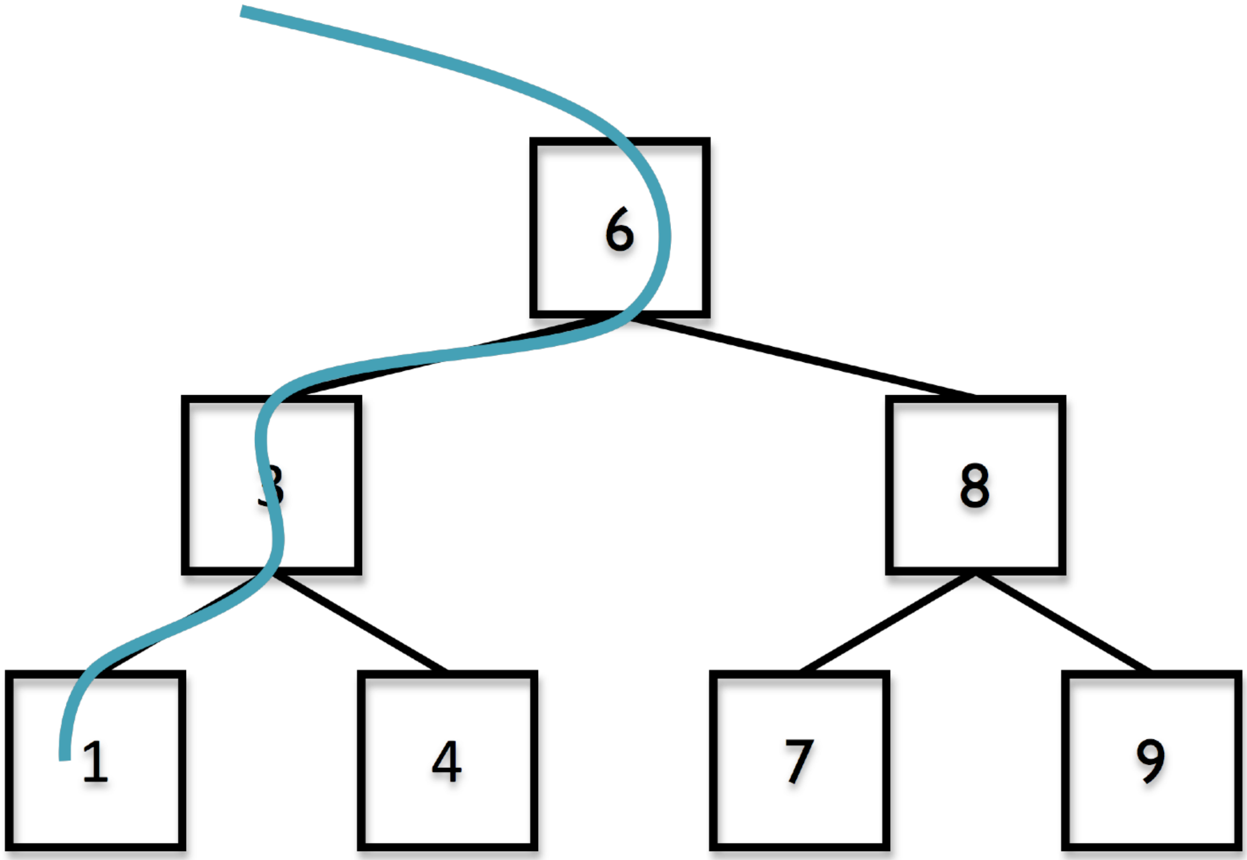
Read-set: 6, 3, 1

Write-set: 1

Thread 2: insert 5

Read-set:

Write-set:



Parallel Programming With TM

Thread 1: insert 2

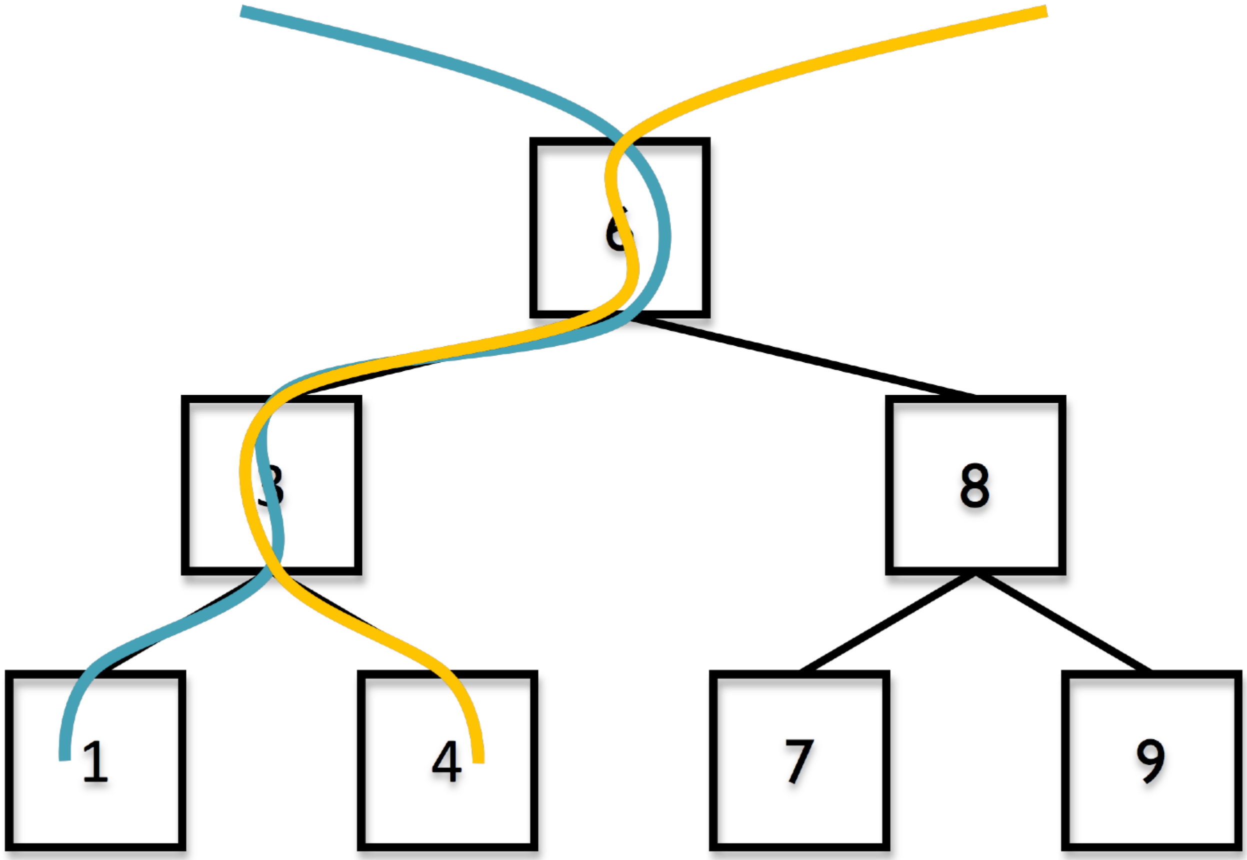
Read-set: 6, 3, 1

Write-set: 1

Thread 2: insert 5

Read-set: 6, 3, 4

Write-set: 4



Parallel Programming With TM

Thread 1: insert 2

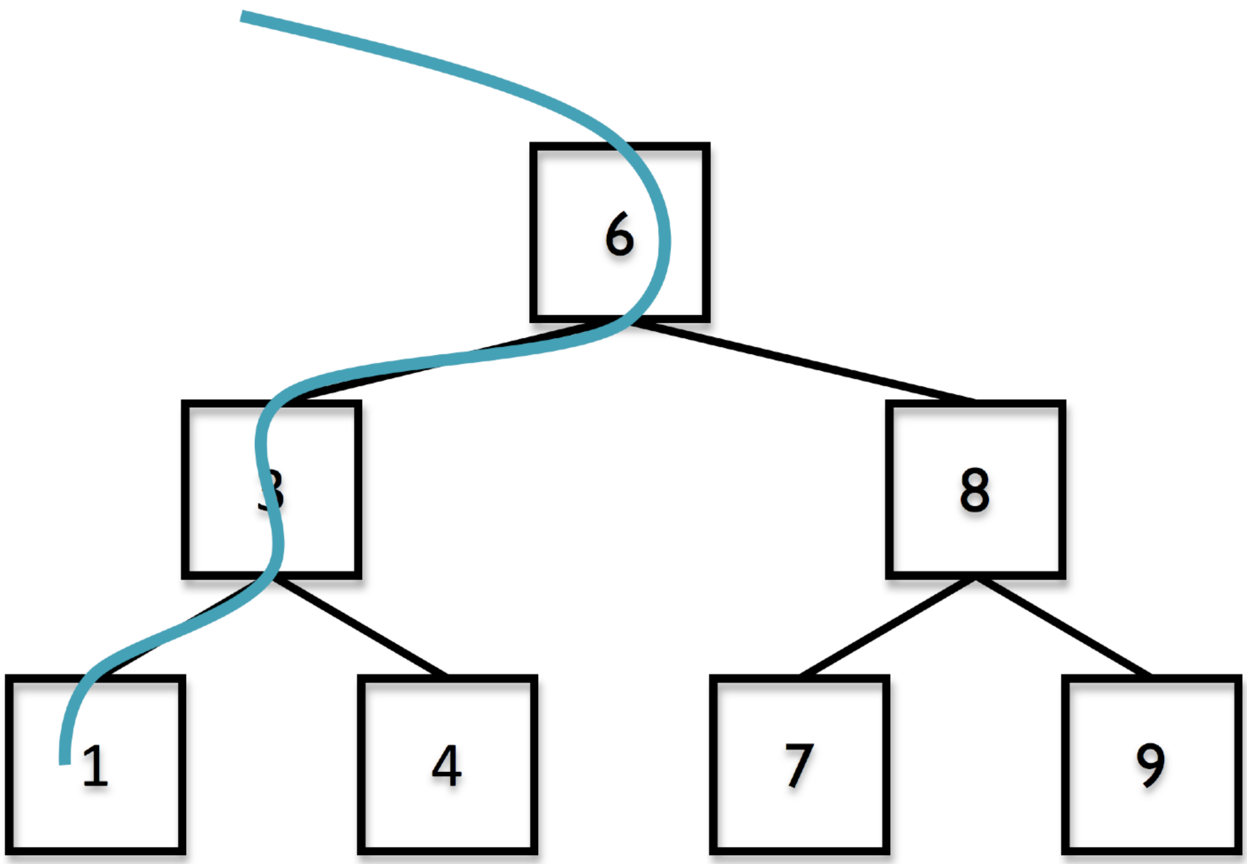
Read-set: 6, 3, 1

Write-set: 1

Thread 2: insert 0

Read-set:

Write-set:



Parallel Programming With TM

Thread 1: insert 2

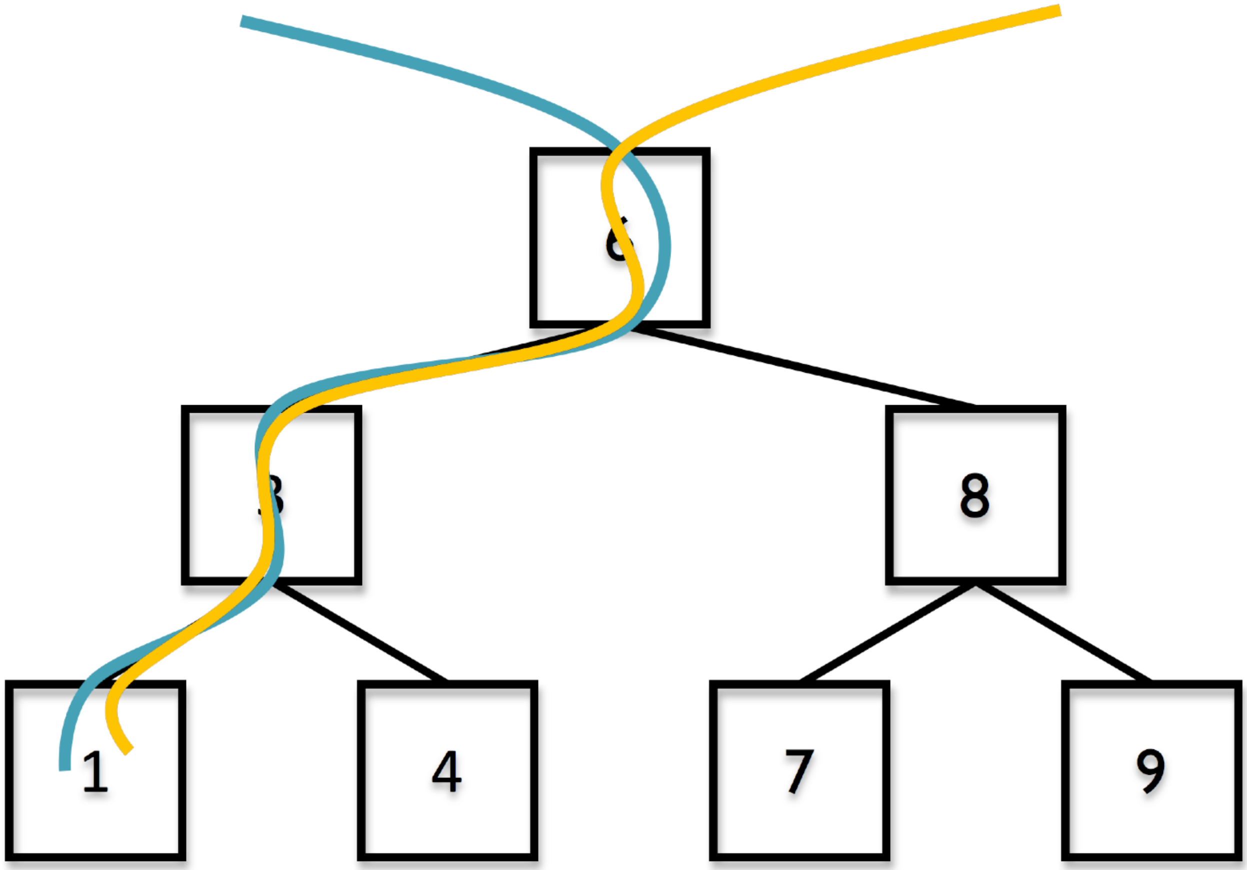
Read-set: 6, 3, 1

Write-set: 1

Thread 2: insert 0

Read-set: 6, 3, 1

Write-set: 1



Outline

- Introduction
- Transactional Memory Primer
- Design of STAMP
- Evaluation of STAMP
- Conclusions

Multiprocessor Benchmarks

- Benchmarks for multiprocessors
  - SPLASH-2 (1995), SPEComp (2001), PARSEC (2008)
  - Not well-suited for evaluating TM
    - Regular algorithms without synchronization problems
    - No annotations for TM

- Benchmarks for TM systems
  - Microbenchmarks from RSTMv3 (2006)
  - STMBench7 (2007)
  - Haskell applications by Perfumo et. al (2007)

TM Benchmark Suite Requirements

- *Breadth:* variety of algorithms & app domains
- *Depth:* wide range of transactional behaviors
- *Portability:* runs on many classes of TM systems

Benchmark	Breadth	Depth	Portability	Comments
RSTMv3	no	yes	yes	Microbenchmarks
STMbench7	no	yes	yes	Single program
Perfumo et al.	no	yes	no	Microbenchmarks; Written in Haskell

STAMP Meets 3 Requirements

- Breadth
  - 8 applications covering different domains & algorithms
  - TM simplified development of each
    - Most not trivially parallelizable
    - Many benefit from optimistic concurrency
- Depth
  - Wide range of important transactional behaviors
    - Transaction length, read & write set size, contention amount
    - Facilitated by multiple input data sets & configurations per app
  - Most spend significant execution time in transactions
- Portability
  - Written in C with macro-based transaction annotations
  - Works with Hardware TM (HTM), Software TM (STM), and hybrid TM

STAMP Applications

Application	Domain	Description
bayes	Machine learning	Learns structure of a Bayesian network
genome	Bioinformatics	Performs gene sequencing
intruder	Security	Detects network intrusions
kmeans	Data mining	Implements K-means clustering
labyrinth	Engineering	Routes paths in maze
ssca2	Scientific	Creates efficient graph representation
vacation	Online transaction processing	Emulates travel reservation system
yada	Scientific	Refines a Delaunay mesh

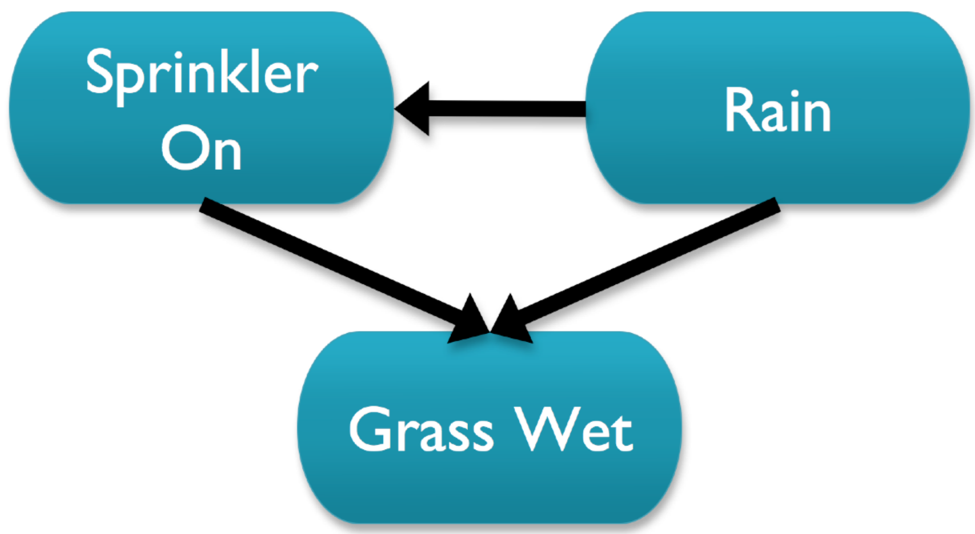
STAMP Applications

Application	Domain	Description
bayes	Machine learning	Learns structure of a Bayesian network
genome	Bioinformatics	Performs gene sequencing
intruder	Security	Detects network intrusions
kmeans	Data mining	Implements K-means clustering
labyrinth	Engineering	Routes paths in maze
ssca2	Scientific	Creates efficient graph representation
vacation	Online transaction processing	Emulates travel reservation system
yada	Scientific	Refines a Delaunay mesh

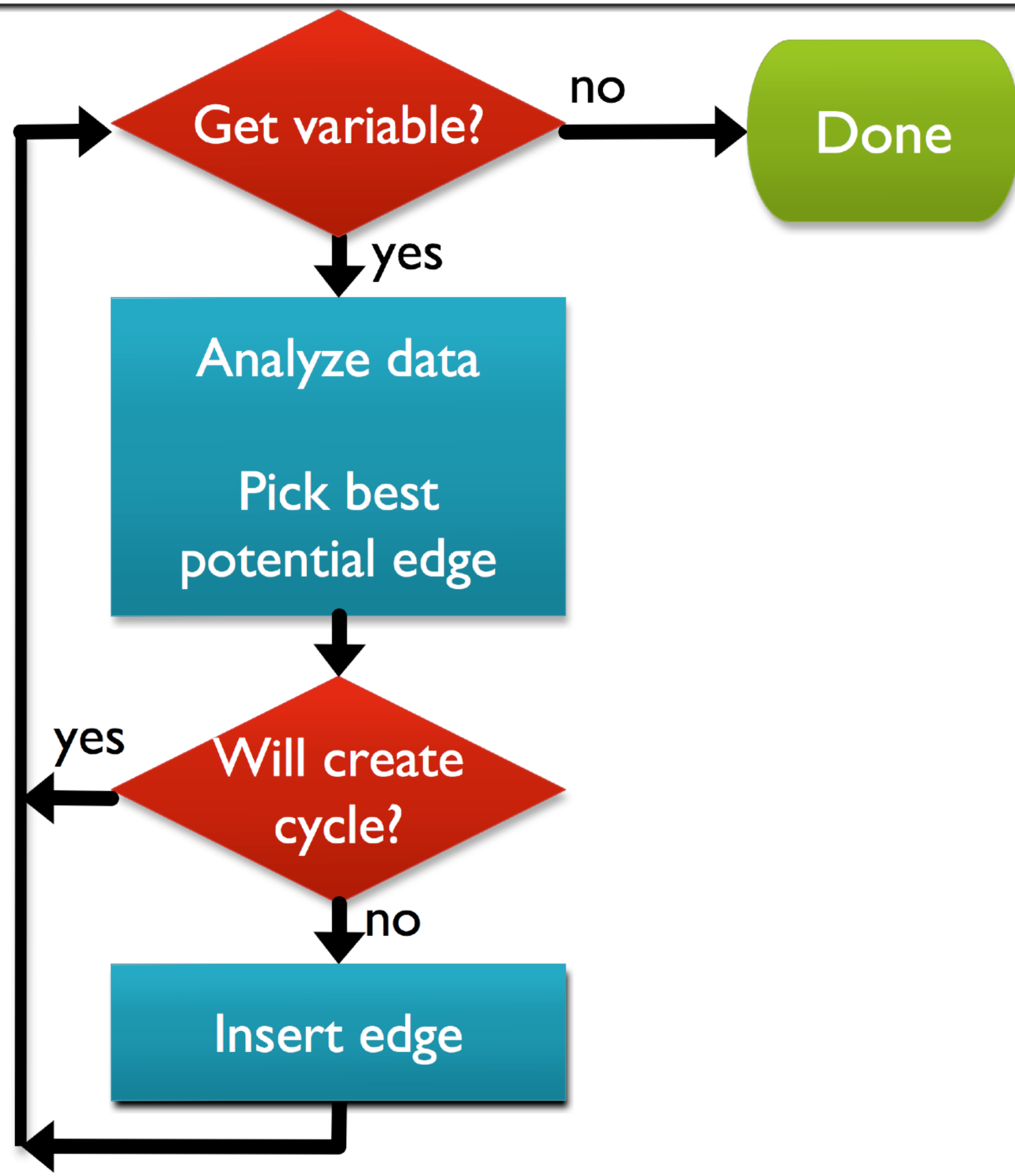
Bayes Description

- Learns relationships among variables from observed data

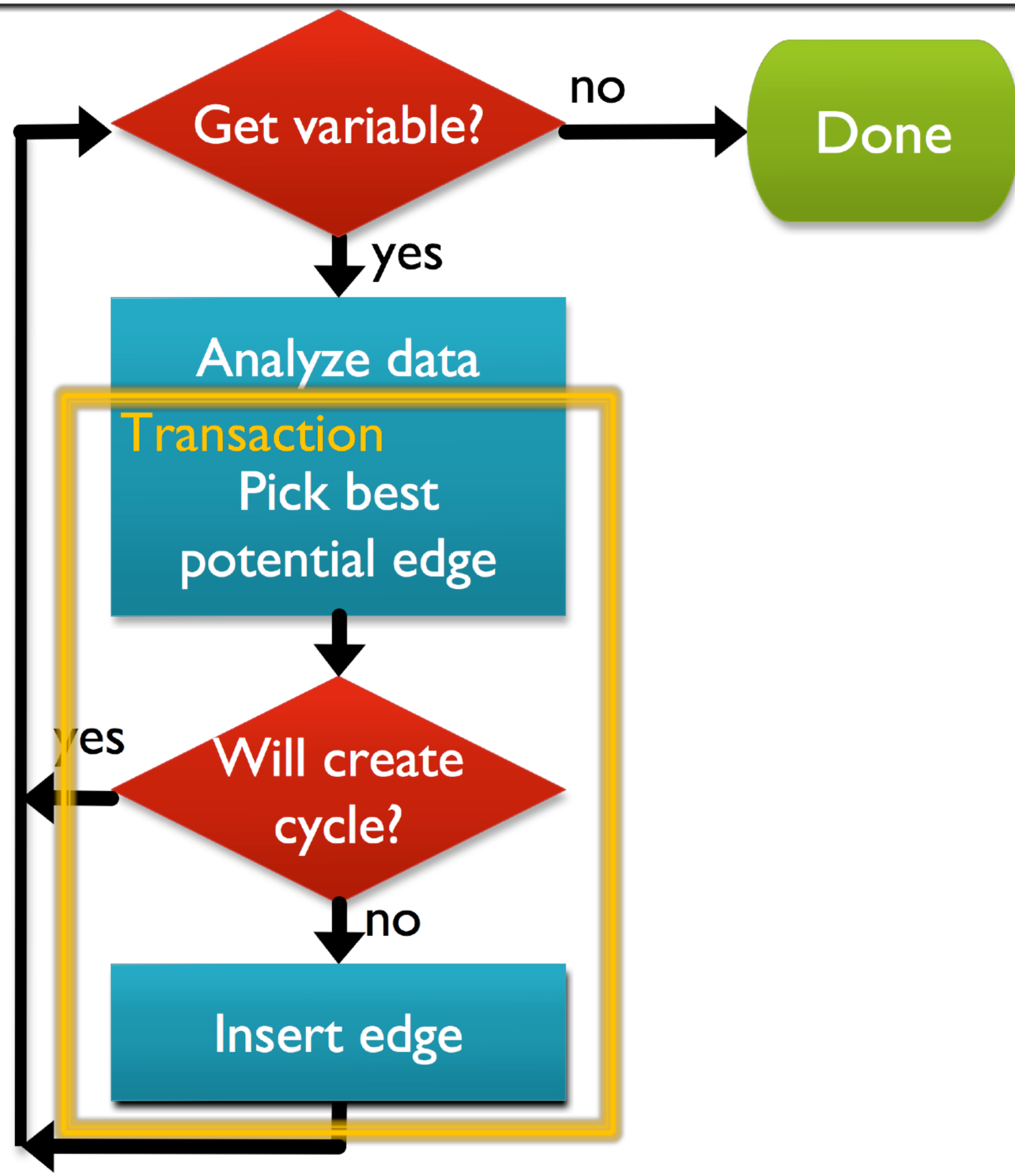
- Relationships are edges in directed acyclic graph:



Bayes Algorithm

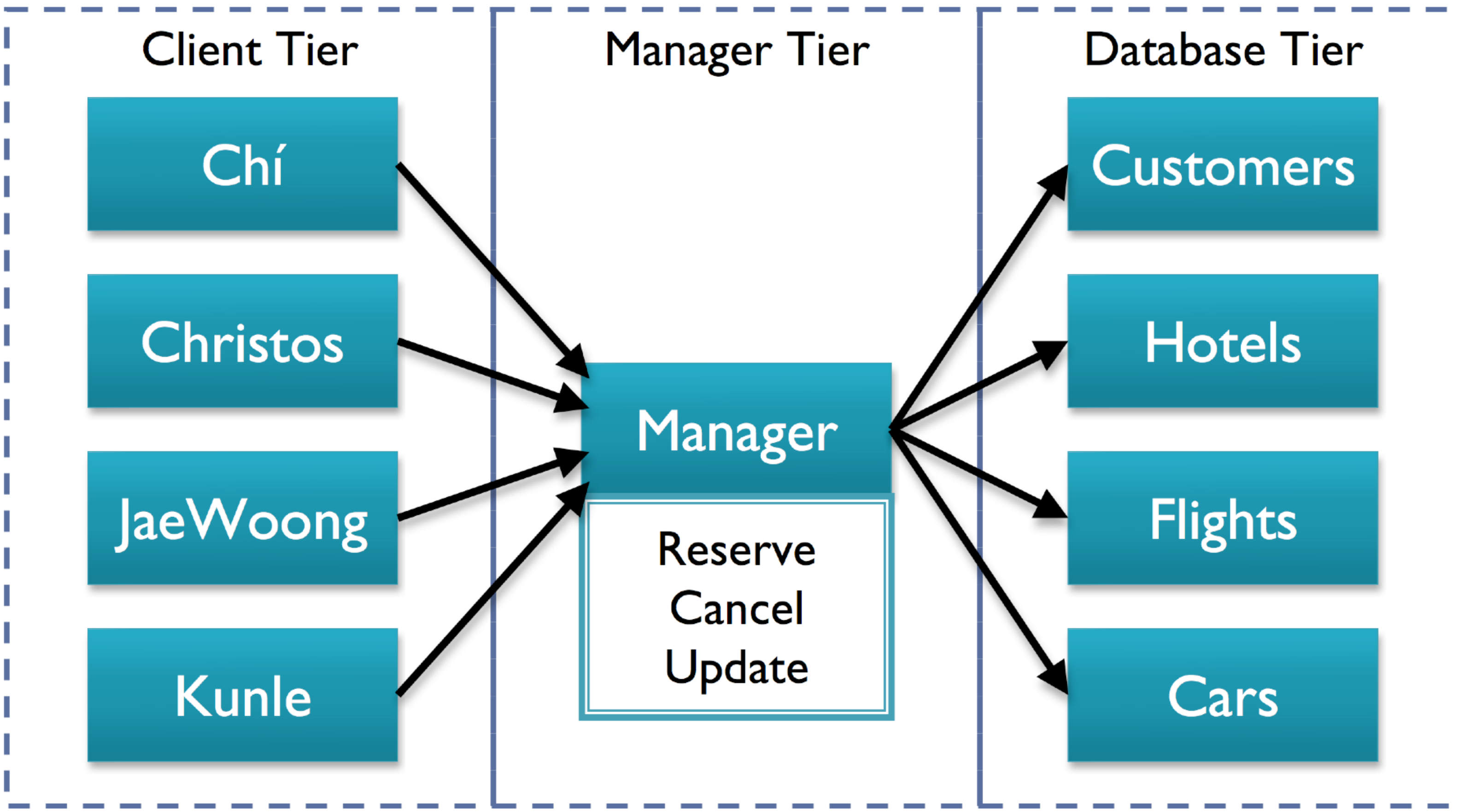


Bayes Algorithm

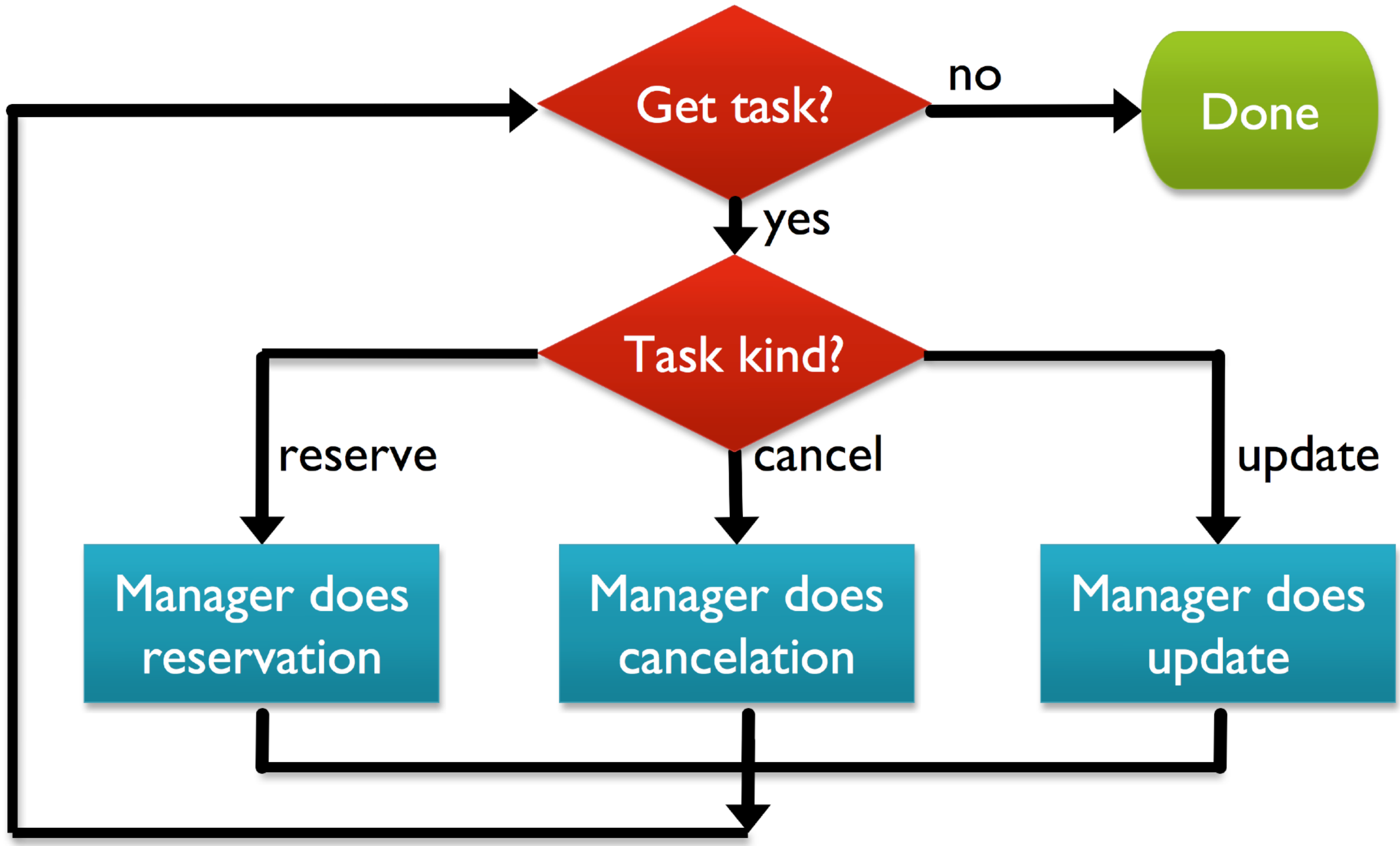


Vacation Description

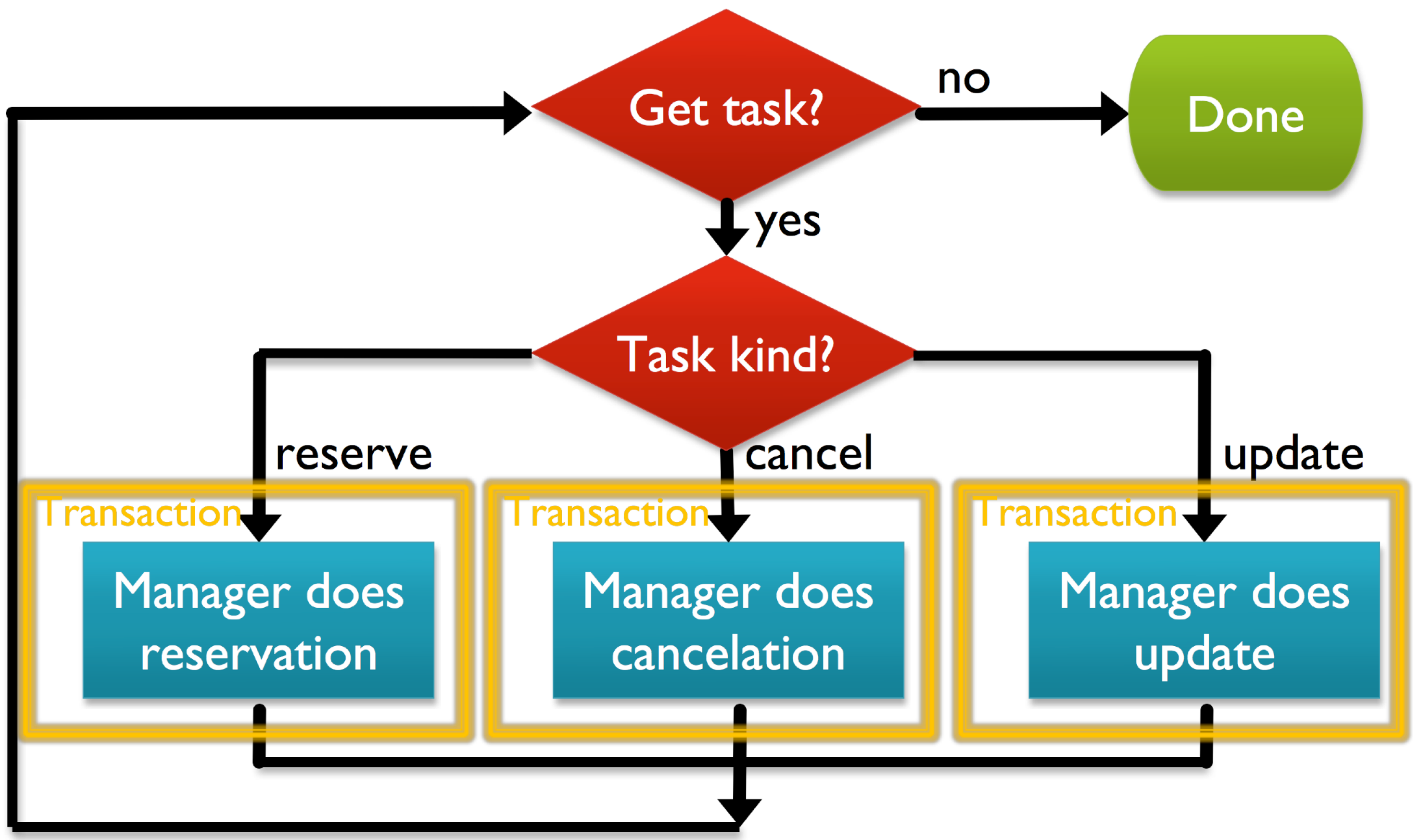
- Emulates travel reservation system
  - Similar to 3-tier design in SPECjbb2000



Vacation Algorithm



Vacation Algorithm



Outline

- Introduction
- Transactional Memory Primer
- Design of STAMP
- Evaluation of STAMP
- Conclusions

Experimental Setup

- Execution-driven simulation
  - 1–16 core x86 chip-multiprocessor with MESI coherence
  - Supports various TM implementations:
    - Hardware TMs (HTMs)
    - Software TMs (STMs)
    - Hybrid TMs
- Ran STAMP on simulated TM systems
- Two experiments:
  - What transactional characteristics are covered in STAMP?
  - Can STAMP help us compare TM systems?

STAMP Characterization

Application	Per Transaction				Time in Transactions
	Instructions	Reads	Writes	Retries	
bayes	60584	24	9	0.59	83%
genome	1717	32	2	0.14	97%
intruder	330	71	16	3.54	33%
kmeans	153	25	25	0.81	3%
labyrinth	219571	35	36	0.94	100%
ssca2	50	1	2	0.00	17%
vacation	3161	401	8	0.02	92%
yada	9795	256	108	2.51	100%

STAMP Characterization

Application	Per Transaction				Time in Transactions
	Instructions	Reads	Writes	Retries	
bayes	60584	24	9	0.59	83%
genome	1717	32	2	0.14	97%
intruder	330	71	16	3.54	33%
kmeans	153	25	25	0.81	3%
labyrinth	219571	35	36	0.94	100%
ssca2	50	1	2	0.00	17%
vacation	3161	401	8	0.02	92%
yada	9795	256	108	2.51	100%

STAMP Characterization

Application	Per Transaction				Time in Transactions
	Instructions	Reads	Writes	Retries	
bayes	60584	24	9	0.59	83%
genome	1717	32	2	0.14	97%
intruder	330	71	16	3.54	33%
kmeans	153	25	25	0.81	3%
labyrinth	219571	35	36	0.94	100%
ssca2	50	1	2	0.00	17%
vacation	3161	401	8	0.02	92%
yada	9795	256	108	2.51	100%

STAMP Characterization

Application	Per Transaction				Time in Transactions
	Instructions	Reads	Writes	Retries	
bayes	60584	24	9	0.59	83%
genome	1717	32	2	0.14	97%
intruder	330	71	16	3.54	33%
kmeans	153	25	25	0.81	3%
labyrinth	219571	35	36	0.94	100%
ssca2	50	1	2	0.00	17%
vacation	3161	401	8	0.02	92%
yada	9795	256	108	2.51	100%

STAMP Characterization

Application	Per Transaction				Time in Transactions
	Instructions	Reads	Writes	Retries	
bayes	60584	24	9	0.59	83%
genome	1717	32	2	0.14	97%
intruder	330	71	16	3.54	33%
kmeans	153	25	25	0.81	3%
labyrinth	219571	35	36	0.94	100%
ssca2	50	1	2	0.00	17%
vacation	3161	401	8	0.02	92%
yada	9795	256	108	2.51	100%

STAMP Characterization

Application	Per Transaction				Time in Transactions
	Instructions	Reads	Writes	Retries	
bayes	60584	24	9	0.59	83%
genome	1717	32	2	0.14	97%
intruder	330	71	16	3.54	33%
kmeans	153	25	25	0.81	3%
labyrinth	219571	35	36	0.94	100%
ssca2	50	1	2	0.00	17%
vacation	3161	401	8	0.02	92%
yada	9795	256	108	2.51	100%

STAMP Characterization

Application	Per Transaction				Time in Transactions
	Instructions	Reads	Writes	Retries	
bayes	60584	24	9	0.59	83%
genome	1717	32	2	0.14	97%
intruder	330	71	16	3.54	33%
kmeans	153	25	25	0.81	3%
labyrinth	219571	35	36	0.94	100%
ssca2	50	1	2	0.00	17%
vacation	3161	401	8	0.02	92%
yada	9795	256	108	2.51	100%

STAMP Characterization

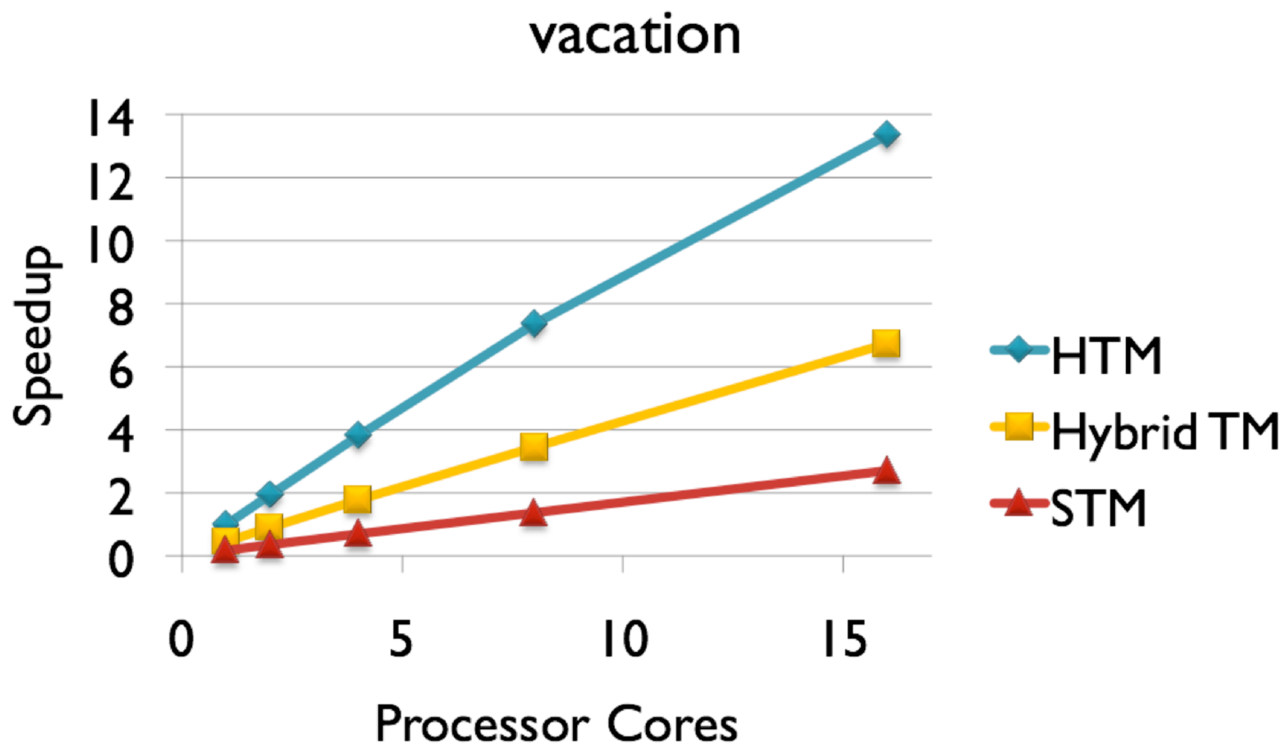
Application	Per Transaction				Time in Transactions
	Instructions	Reads	Writes	Retries	
bayes	60584	24	9	0.59	83%
genome	1717	32	2	0.14	97%
intruder	330	71	16	3.54	33%
kmeans	153	25	25	0.81	3%
labyrinth	219571	35	36	0.94	100%
ssca2	50	1	2	0.00	17%
vacation	3161	401	8	0.02	92%
yada	9795	256	108	2.51	100%

STAMP Characterization

Application	Per Transaction				Time in Transactions
	Instructions	Reads	Writes	Retries	
bayes	60584	24	9	0.59	83%
genome	1717	32	2	0.14	97%
intruder	330	71	16	3.54	33%
kmeans	153	25	25	0.81	3%
labyrinth	219571	35	36	0.94	100%
ssca2	50	1	2	0.00	17%
vacation	3161	401	8	0.02	92%
yada	9795	256	108	2.51	100%

Using STAMP to Compare TMs (1)

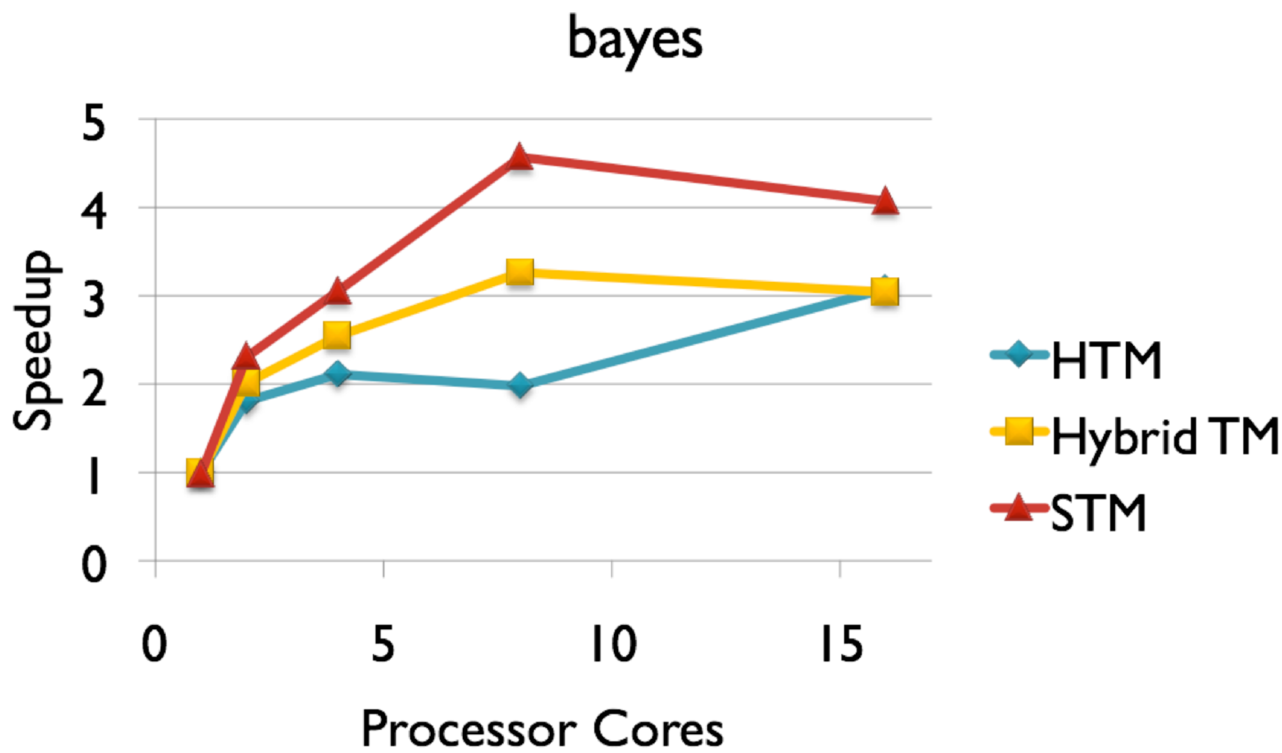
- Measured speedup on 1–16 cores for various TMs



- In general, hybrid faster than STM but slower than HTM

Using STAMP to Compare TMs (2)

- Sometimes the behavior is different from anticipated



- Lesson: Importance of conflict detection granularity

Using STAMP to Compare TMs (3)

- Some other lessons we learned:
  - Importance of handling very large read & write sets (labyrinth)
  - Optimistic conflict detection helps forward progress (intruder)
- Diversity in STAMP allows thorough TM analysis
  - Helps identify (sometimes unexpected) TM design shortcomings
  - Motivates directions for further improvements
- STAMP can be a valuable tool for future TM research

Conclusions

- STAMP is a comprehensive benchmark suite for TM
  - Meets *breadth*, *depth*, and *portability* requirements
  - Useful tool for analyzing TM systems

- Public release: `http://stamp.stanford.edu`
  - Early adopters:
    - Industry: Microsoft, Intel, Sun, & more
    - Academia: U. Wisconsin, U. Illinois, & more
  - TL2-x86 STM

Questions?

http://stamp.stanford.edu