Jin-Soo Kim
(*jinsoo.kim@snu.ac.kr*)

Systems Software &
Architecture Lab.

Seoul National University

Fall 2020

Lottery Scheduling

(Carl Waldspurger et al., OSDI '94)

Priority-based Scheduling Schemes

- The notion of priority does not provide the encapsulation and modularity properties
- The assignment of priorities and dynamic priority adjustment schemes are ad-hoc
  - Adjusting scheduling parameters is at best a black art
- Poorly understood
- Schedulers are complex and difficult to control
- Priority inversion problem
- Fair share schedulers are implemented by adjusting priorities with a feedback loop (relatively coarse control over long-running applications)

Goals

- Flexible and responsive control over the relative execution rates of computations
  - Rapid, dynamic control over scheduling at a time scale of milliseconds to seconds
- Proportional sharing
  - The resource consumption rates of active computations are proportional to the relative shares that they are allocated
- Support for modular resource management
  - Can be generalized to manage other resources, such as I/O bandwidth, memory, and access to locks
- Simple and efficient implementation

Lottery Scheduling and Tickets

- A randomized resource allocation mechanism based on tickets and lotteries

- Tickets
  - Encapsulate abstract, relative, and uniform resource rights
  - Abstract: they quantify resource rights independently of machine details
  - Relative: the fraction of a resource that they represent varies dynamically in proportion to the contention for that resource
  - Uniform: rights for heterogeneous resources can be homogeneously represented as tickets

  - Similar to the properties of money

Lotteries

- Scheduler picks the winning ticket randomly, and gives the owner the resource
- Probabilistically fair
  - The expected allocation of resources is proportional to the number of tickets that they hold
- The scheduling algorithm is randomized
  - The actual allocated proportional are not guaranteed to match the expected proportions exactly
  - The disparity between them decreases as the number of allocations increases

Performance Characteristics

- The number of lotteries won by a client:
  - Binomial distribution
  - The winning probability *p* (total *T* tickets): $p = t/T$
  - The expected number of wins *w* after *n* lotteries:

$$E[w] = np \qquad \sigma_w^2 = np(1-p) \qquad \sigma_w/E[w] = \sqrt{(1-p)/np}$$

  - A client's throughput is proportional to its ticket allocation
- The number of lotteries required for a client's first win:
  - Geometric distribution
  - The expected number of lotteries *n* that a client must wait before its first win:

$$E[n] = 1/p \qquad \sigma_n^2 = (1-p)/p^2$$

  - The client's average response time is inversely proportional to its ticket allocation

Performance Characteristics (cont'd)

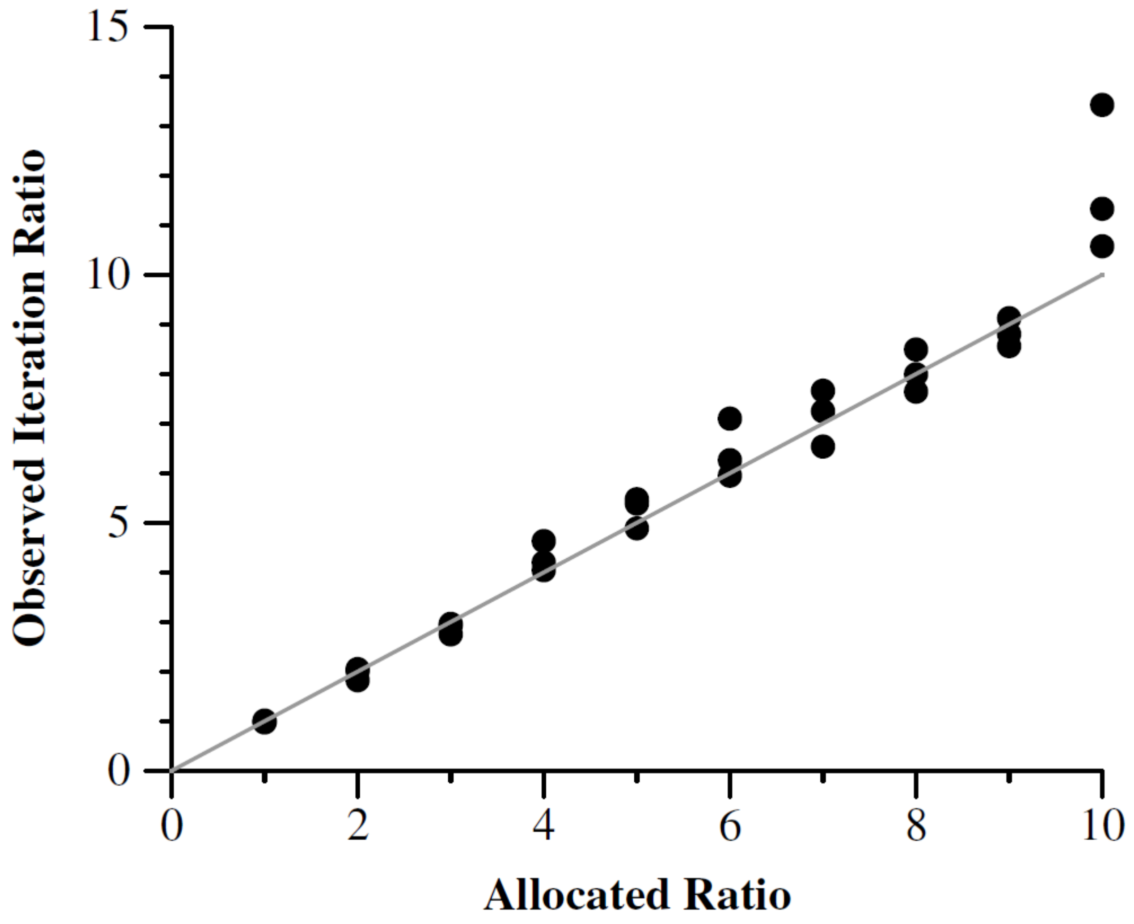
- The accuracy improves with $\sqrt{n}$
  - Need frequent lotteries
  - With a scheduling quantum of 10 msec (100 lotteries/sec), reasonable fairness can be achieved over sub-second time intervals
  - Mostly accurate, but short-term inaccuracies are possible
- No starvation
  - Any client with a non-zero number of tickets will eventually win a lottery
- Responsive
  - Any changes to relative ticket allocations are immediately reflected in the next lottery

Example: Fairness

- Two Dhrystone (CPU-intensive) benchmark tasks for 60 sec.

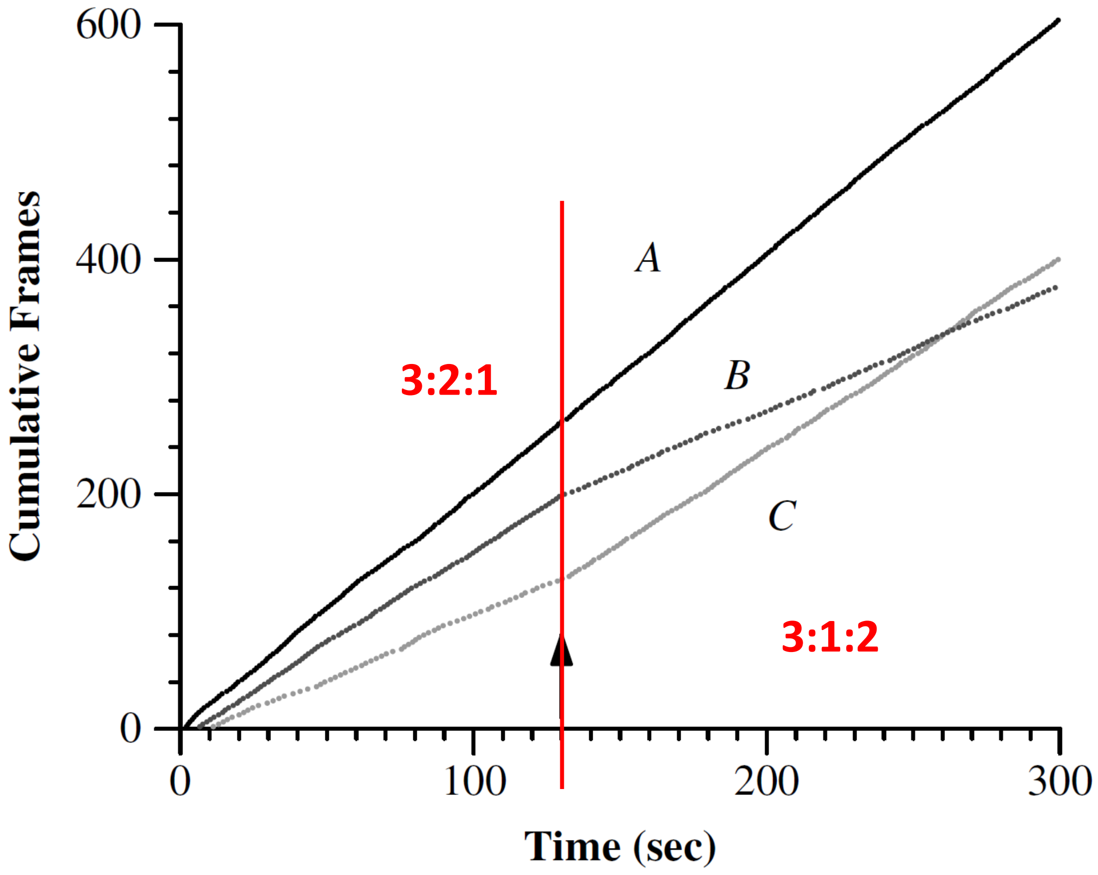
- The variance is greater for larger ratios:
  - 13.42 : 1 (for 10 : 1)

- Even larger ratios converge over longer time intervals:
  - 19.08 : 1 (for 20 : 1, for 3 min.)



Example: Multimedia Applications

- Three mpeg_play video viewers
- Not exact
  - 1.92 : 1.50 : 1 (3 : 2 : 1)
  - 1.92 : 1 : 1.53 (3 : 1 : 2)
- Due to the round-robin processing of client requests by the single-threaded X11R5 server
  - 3.06 : 2.04 : 1 with `-no-display` option



Compensation Tickets

- What happens if a thread is I/O-bound and blocks before its quantum expires?
  - The thread gets less than its share of the processor

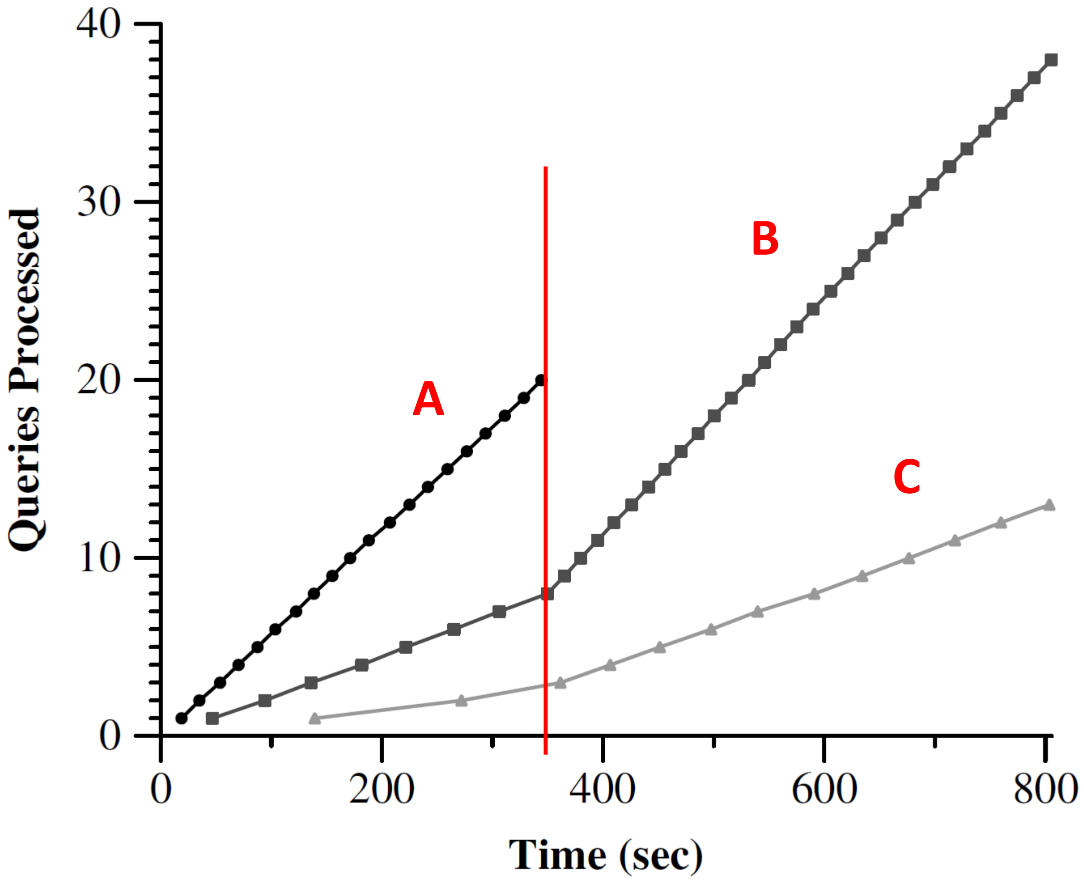
- If a thread consumes only a fraction *f* of the quantum, its tickets are inflated by 1/*f* until the next time you win
  - If A on average uses 1/5 of a quantum, its tickets will be inflated 5x and it will win 5 times as often and get its correct share overall

Ticket Transfer

- If you are blocked on someone else, give them your tickets
- Useful for client-server system
  - Server has no tickets of its own
  - Clients give their tickets to server threads during RPC
  - Server's priority is the sum of the priorities of all of its active clients
  - Server can use lottery scheduling to give preferential service to high-priority clients
  - Clients also have the ability to divide ticket transfers across multiple servers on which they may be waiting
- Avoid priority inversion problem

Ticket Transfer: Example

- Three clients (8 : 3 : 1 allocations) compete for service from a multithreaded database server (server has no tickets)
- Search a substring over the entire Shakespeare's plays (No I/O)
- Throughput
  - 20 vs. 10 queries (A : B + C)
  - 39 vs. 13 queries (B : C)
- Response time
  - 7.69 : 2.51 : 1 (A : B : C)
  - 2.91 : 1 (B : C)

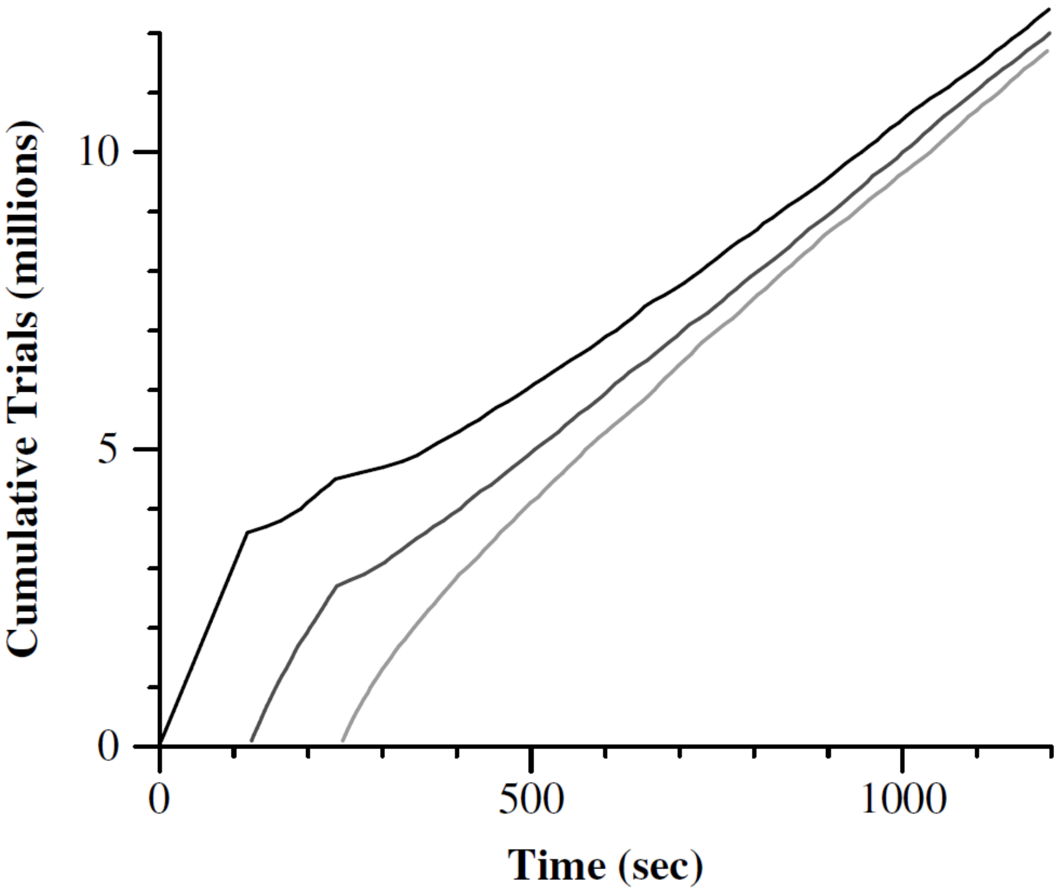


Ticket Inflation

- Make up your own tickets (print your own money)
- Only works among mutually trusting clients
  - Why?
- Presumably works best if inflation is temporary
- Allows clients to adjust their resource allocations without explicit communication
- Examples
  - Monte-Carlo algorithm: dynamically adjust the number of tickets as a function of its current relative error
  - Graphics-intensive programs: a large share to display a crude outline initially, and then a smaller share to compute details

Ticket Inflation: Example

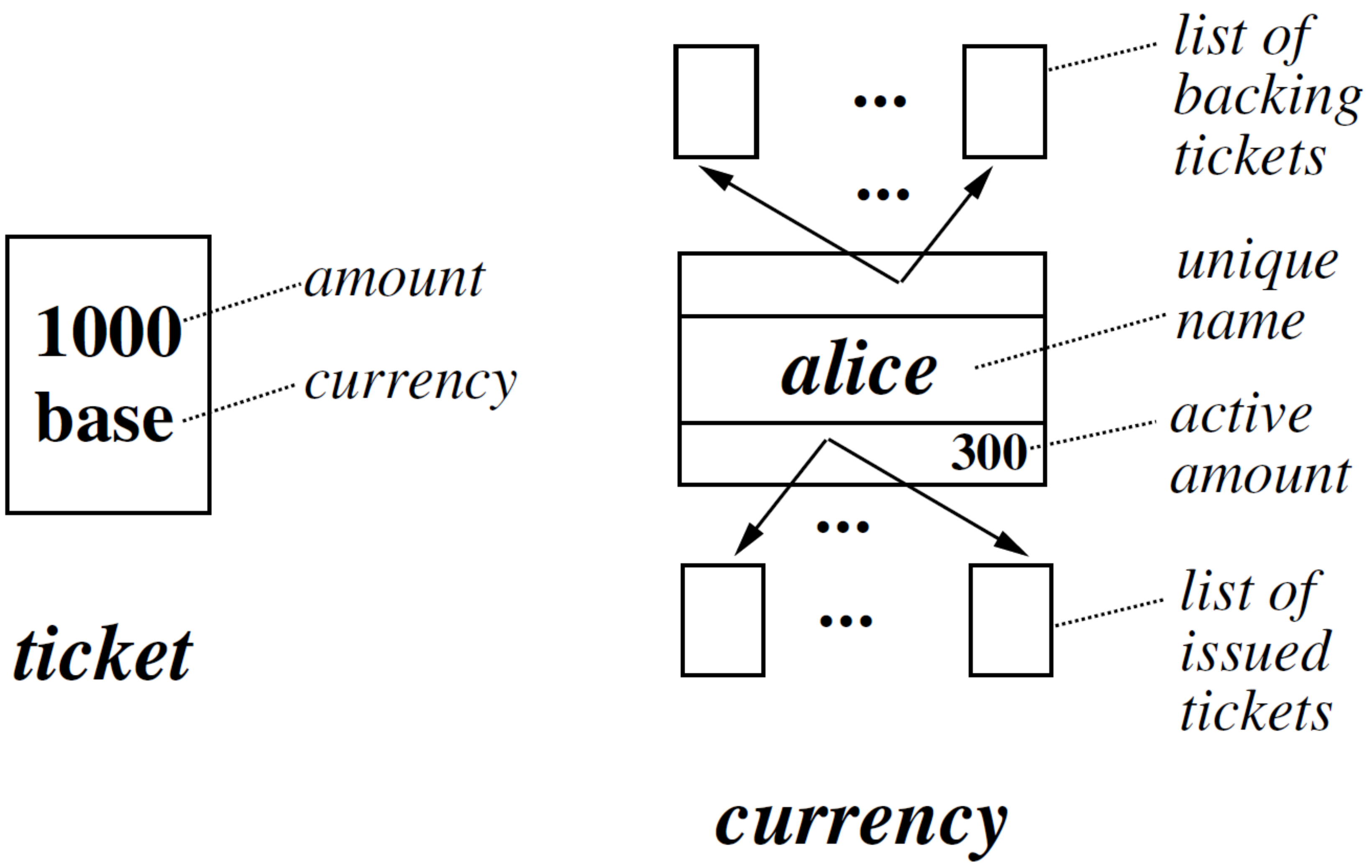
- Three Monte-Carlo tasks
- Ticket inflation proportional to the square of its relative error
- A new task initially receives a large share of the processor



Ticket Currencies

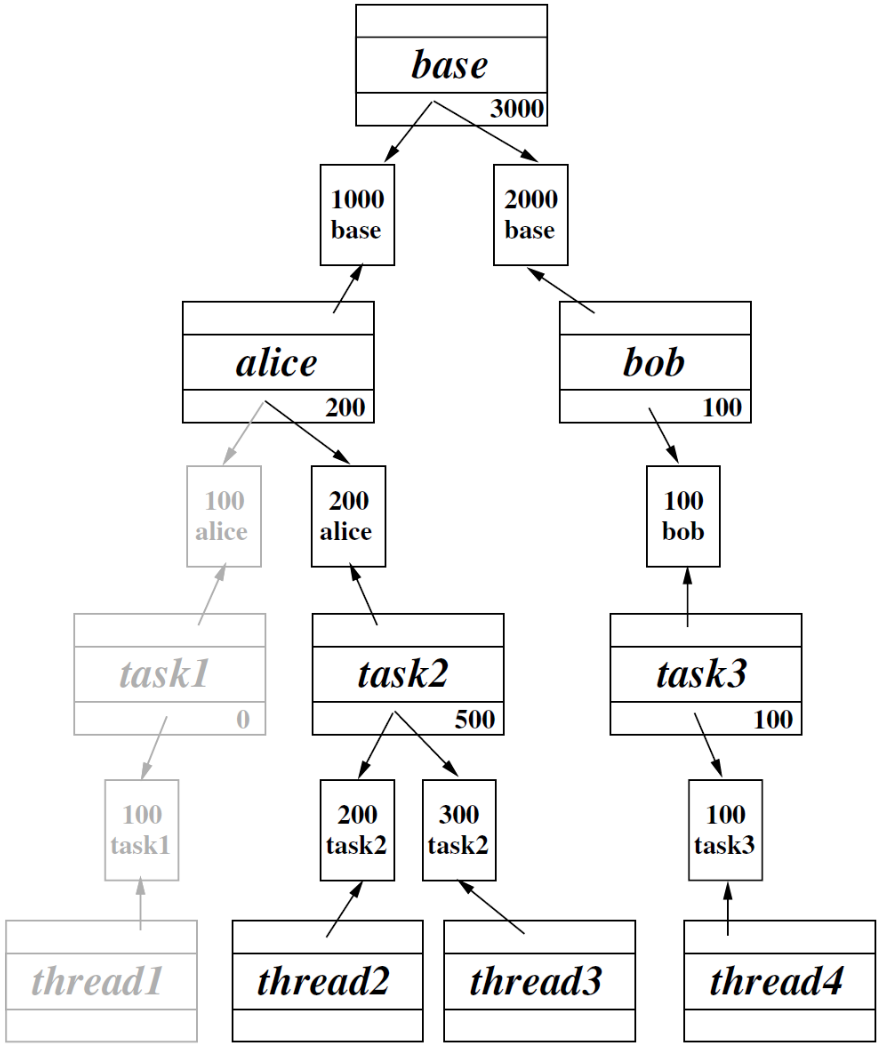
- Express resource rights in units that are local to each group of mutually trusting clients
- A unique currency is used to denominate tickets within each trust boundary
  - Each currency is backed, or funded, by tickets that are denominated in more primitive currencies
  - The effects of inflation can be locally contained by maintaining an exchange rate
- Useful for flexible naming, sharing, and protecting resource rights
  - Simplify mini lottery like mutex inside a group
  - Support fine-grain allocation decisions

Ticket Currencies: Ticket vs. Currency



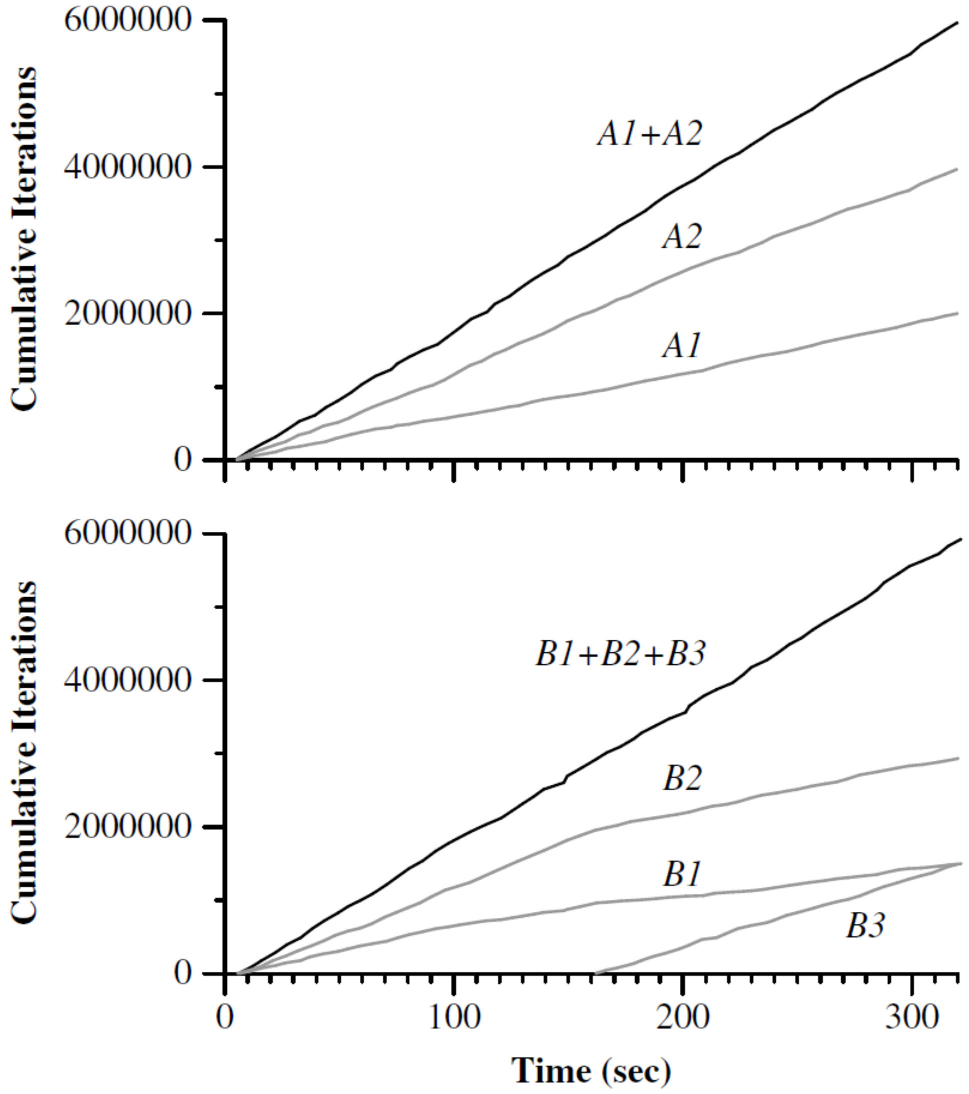
Ticket Currencies: Implementation

- Each current maintains an active amount of sum for all of its issued tickets
- A ticket is active while it is being used by a thread to compete in a lottery
- If a ticket deactivation(activation) changes a currency's active amount to(from) zero, the deactivation(activation) propagates to each of its backing tickets
- Currency relationships may form an acyclic graph



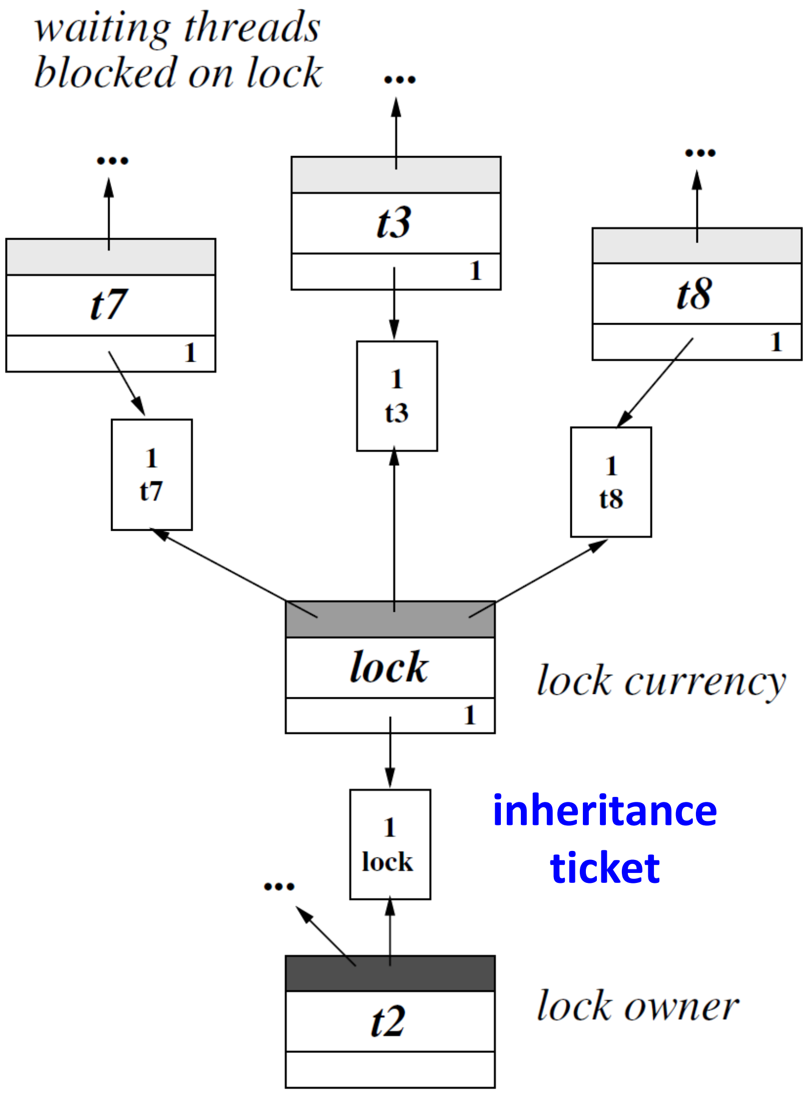
Ticket Currencies: Load Insulation

- 5 Dhrystone tasks
- Two currencies A and B
  - Funded equally
- Task group A
  - A1 with 100.A
  - A2 with 200.A
- Task group B
  - B1 with 100.B
  - B2 with 200.B
  - Later, added B3 with 300.B



Ticket Currencies: Lock Funding

- A lottery-scheduled mutex has
  - Mutex currency
  - Inheritance ticket
- Waiting threads fund the mutex currency
  - The mutex transfers its inheritance ticket to the thread which currently holds the mutex
- When done, mutex holder conducts a lottery to determine the next holder
  - Passes on inheritance ticket

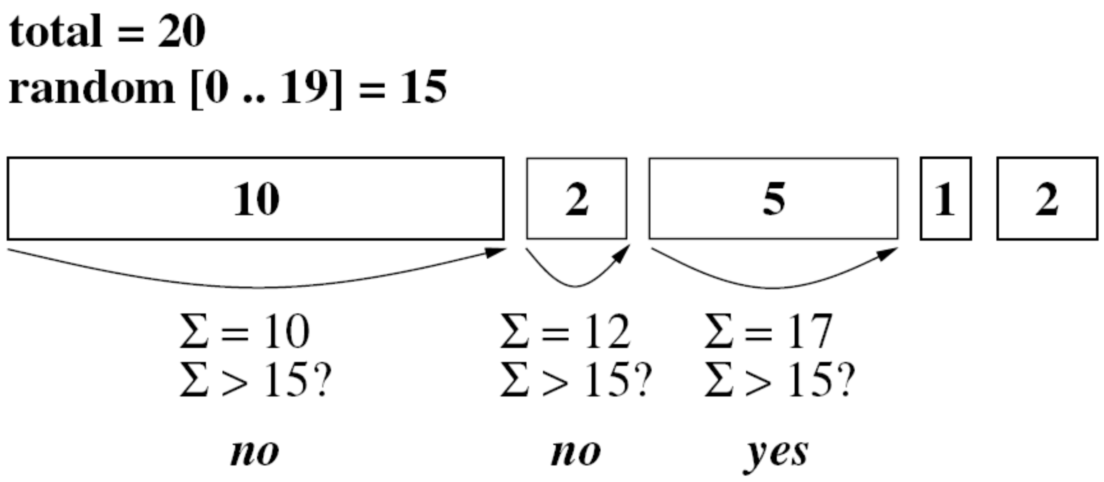


Implementation Issues

- Frequent lotteries require efficiency

- A fast random number generator
  - Based on Park-Miller algorithm
  - Executed in about 10 RISC instructions

- Fast selection of ticket based on random number
  - Straightforward algorithm: O(n)
    - Clients may be ordered by decreasing ticket counts
  - Tree-based algorithm: O(log n)
    - Uses a tree of partial ticket sums, with clients at the leaves



Discussion

- Not as fair as we expected
  - Mutex comes out 1.8 : 1 instead of 2 : 1
  - Multimedia applications come out 1.92 : 1.50 : 1 instead of 3 : 2 : 1
- To really work, tickets must be used everywhere
  - The results for multimedia applications are distorted due to X server assuming uniform priority instead of using tickets
  - In every queue, spinlock, etc.
- Is there any way to game the scheduler?
- What about kernel cycles?
- Other problems?

Stride Scheduling

(Carl Waldspurger et al., '95)

Stride Scheduling

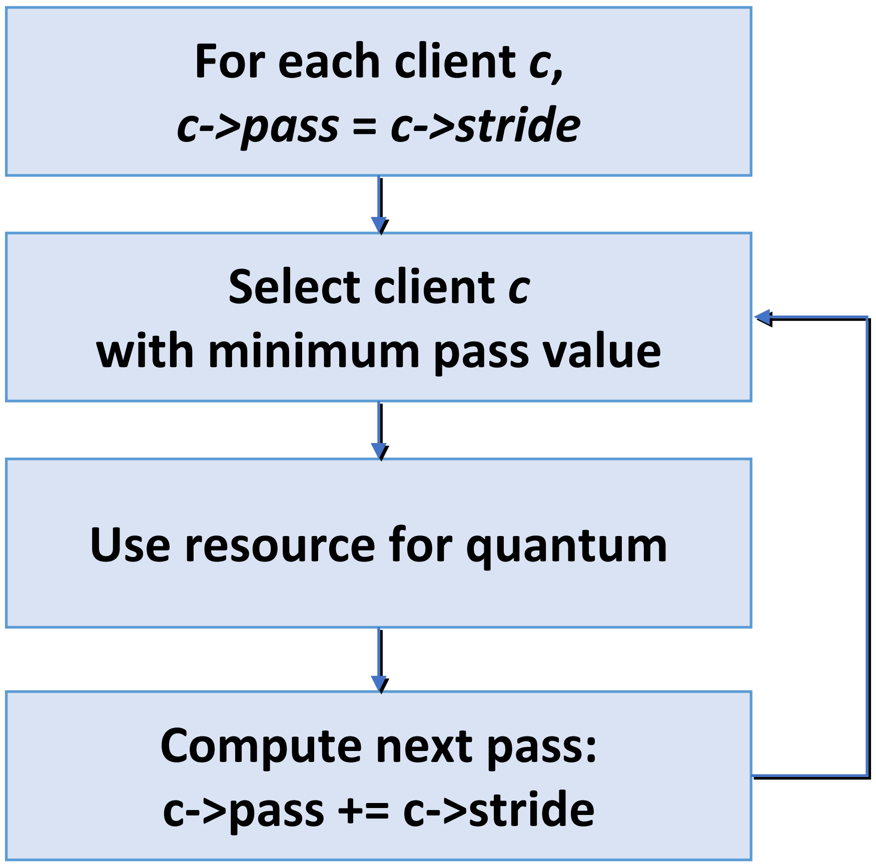
- A deterministic version to reduce short-term variability
  - Based on rate-based flow control algorithms

- Stride
  - The time interval that a client must wait between successive allocations
  - Inversely proportional to the number of ticket
  - High priority jobs have low strides and thus run often
  - Represented in virtual time units called passes

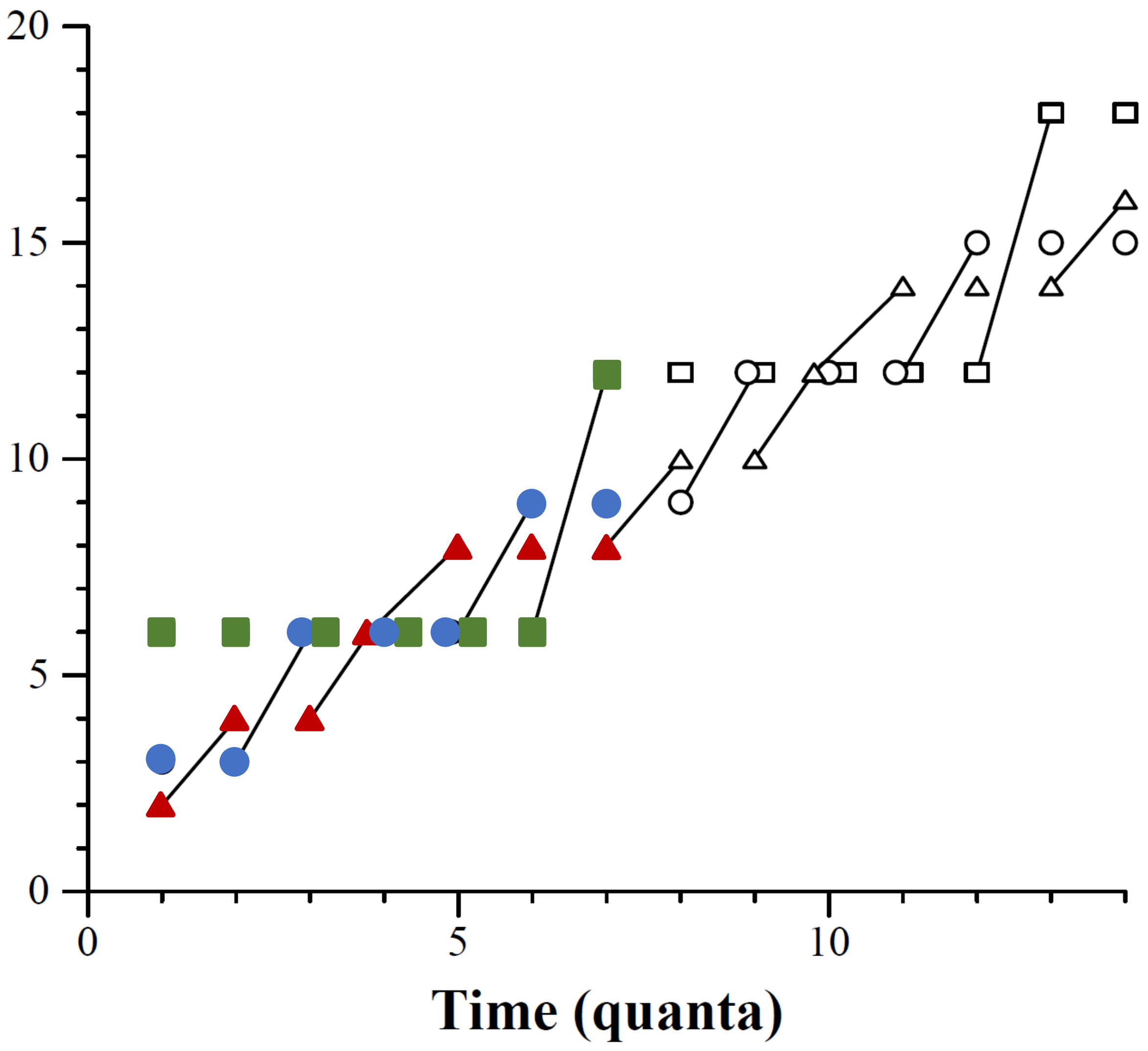
- Result
  - Far more accurate than lottery scheduling
  - Error can be bounded absolutely instead of probabilistically

Basic Algorithm

- Tickets
  - Relative resource allocation
- Strides = $stride_1$ / *tickets*
  - Interval between selection
  - $stride_1$: some large integer constant
- Pass
  - Virtual time index for the next selection
  - Advanced by the client's stride



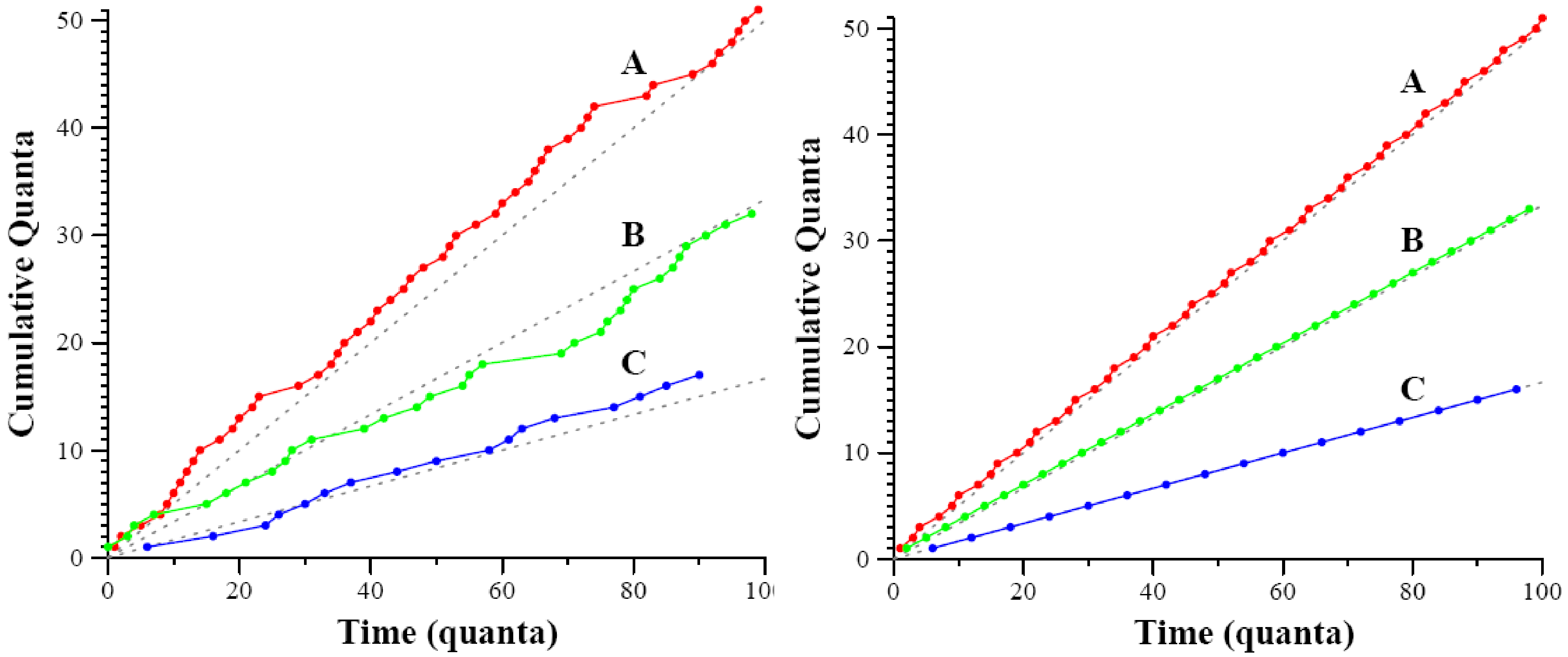
Stride Scheduling: Example



	A ▲	B ●	C ■
Tickets	3	2	1
Strides	2	3	6
Time = 1	2 (+2)	3	6
Time = 2	4	3 (+3)	6
Time = 3	4 (+2)	6	6
Time = 4	6 (+2)	6	6
Time = 5	8	6 (+3)	6
Time = 6	8	9	6 (+6)
Time = 7	8	9	12

Lottery vs. Stride

- Resource Allocations
  - A : B : C = 3 : 2 : 1



Other Issues (See paper)

- Dynamic join and leave
- Dynamic ticket modification
- Compensation tickets?
- Hierarchical stride scheduling

Summary: Lottery vs. Stride

- Probabilistic vs. deterministic

- Stride scheduling
  - Improved accuracy over relative throughput rates, with significant less response time variability
  - Careful state updates are required for dynamic changes

- Lottery scheduling
  - Conceptually simpler than stride scheduling
  - Stateless