Outline for Today

- Objectives:
  - Interrupts (continued)
  - Lottery Scheduling
- Announcements

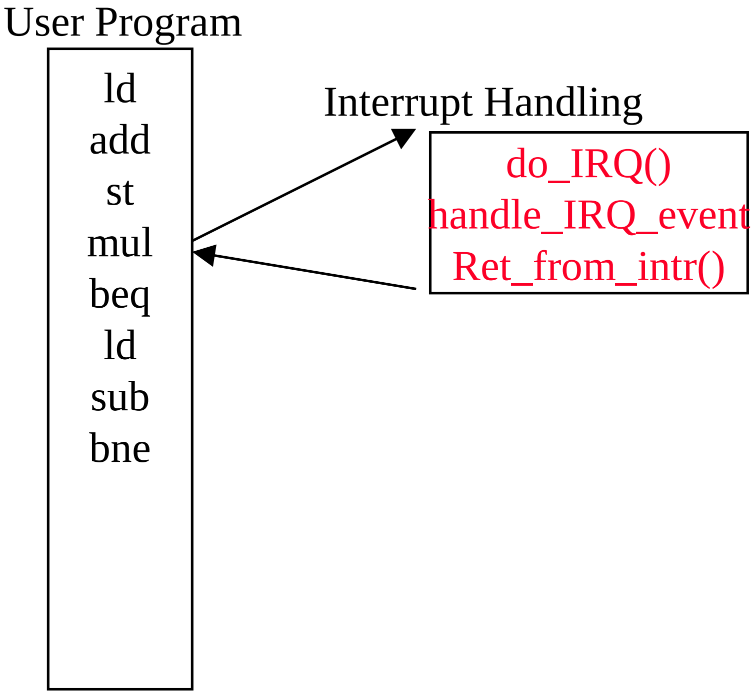
Role of Interrupts in I/O

So, the program needs to access an I/O device…

- Start an I/O operation (special instructions or memory-mapped I/O)
- Device controller performs the operation asynchronously (in parallel with) CPU processing (between controller's buffer & device).
- If DMA, data transferred between controller's buffer and memory without CPU involvement.
- Interrupt signals I/O completion when device is done.

CPU handles Interrupt

- Device raises interrupt line, CPU detects this, CPU stops current operation, disables interrupts, enters kernel mode, saves current program counter, jumps to predefined location, saves other processor state needed to continue at interrupted instruction.
- For each interrupt number, jumps to address of appropriate interrupt service routine. [do_IRQ()]
  - Interrupt context
  - Kernel stack of whatever was interrupted
  - Can not sleep
- Handlers on this line do what needs to be done. [handle_IRQ_event()]
  - Unless SA_INTERRUPT specified when registered, re-enable interrupts during handler execution
  - If line is shared, loop through all handlers
- Restores saved state at interrupted instruction [ret_from_intr()],
  returns to user mode.



User Program

ld
add
st
mul
beq
ld
sub
bne

Interrupt Handling

do_IRQ()
handle_IRQ_event
Ret_from_intr()

Interrupt Control

- local_irq_disable() and local_irq_enable() -- affecting all interrupts for this processor
- local_irq_save(…) and local_irq_restore(…) – save and disable interrupts on this processor and restore previous interrupt state
- disable_irq(irq), disable_irq_nosynch(irq), enable_irq(irq) – affecting particular interrupt line
- Informational:
  - irqs_disabled() – local interrupts disabled?
  - in_interrupt() – in interrupt context or process context?
  - In_irq() – executing an interrupt handler?

Bottom Half Processing

- Deferring work that is too heavyweight for interrupt handling
- Mechanisms:
  - Softirqs
    "soft interrupts", statically defined (32 max.) action functions that can run concurrently on SMP
    pending when bit set in 32-bitmask (usually set in associated interrupt handler [raise_softirq()]
    run with interrupts enabled, proper locking required
  - Tasklets
    dynamically created functions that are built upon softirqs,
    two of the same type can not run concurrently
    lists of tasklet_struct hooked to 2 of the softirqs
  - Workqueue
    implemented as kernel-based "worker" threads with process context of their own -- thus allowed to sleep

Lottery Scheduling

Waldspurger and Weihl (OSDI 94)

Claims

- Goal: responsive control over the relative rates of computation
- Claims:
  - Support for modular resource management
  - Generalizable to diverse resources
  - Efficient implementation of proportional-share resource management: consumption rates of resources by active computations are proportional to relative shares allocated

Basic Idea

- Resource rights are represented by lottery tickets
  - abstract, relative (vary dynamically wrt contention), uniform (handle heterogeneity)
  - responsiveness: adjusting relative # tickets gets immediately reflected in next lottery
- At allocation time: hold a lottery; Resource goes to the computation holding the winning ticket.

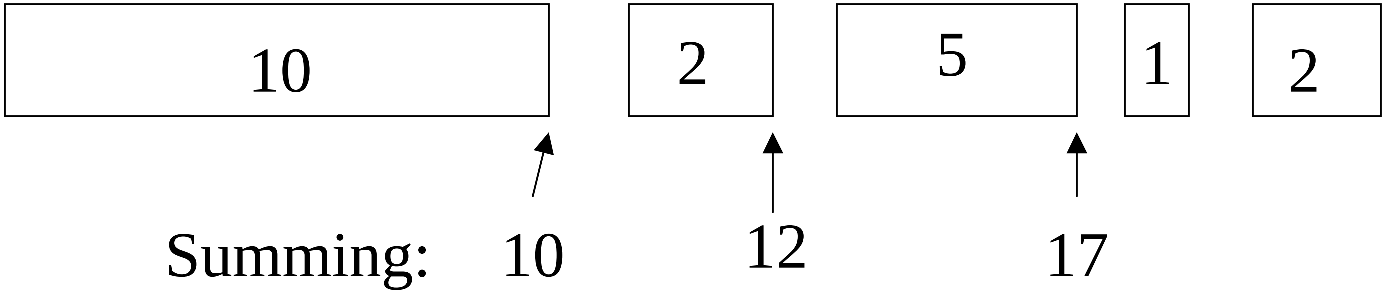
Fairness

- Expected resource allocation is proportional to # tickets held - actual allocation becomes closer over time.
- Throughput – Expected number of lotteries won by client
  $E[w] = n\ p$ where $p = t/T$
- Response time -- # lotteries to wait for first win
  $E[n] = 1/p$
- No starvation

w	# wins
t	# tickets
T	total # tickets
n	# lotteries

Example List-based Lottery

T = 20



10 2 5 1 2

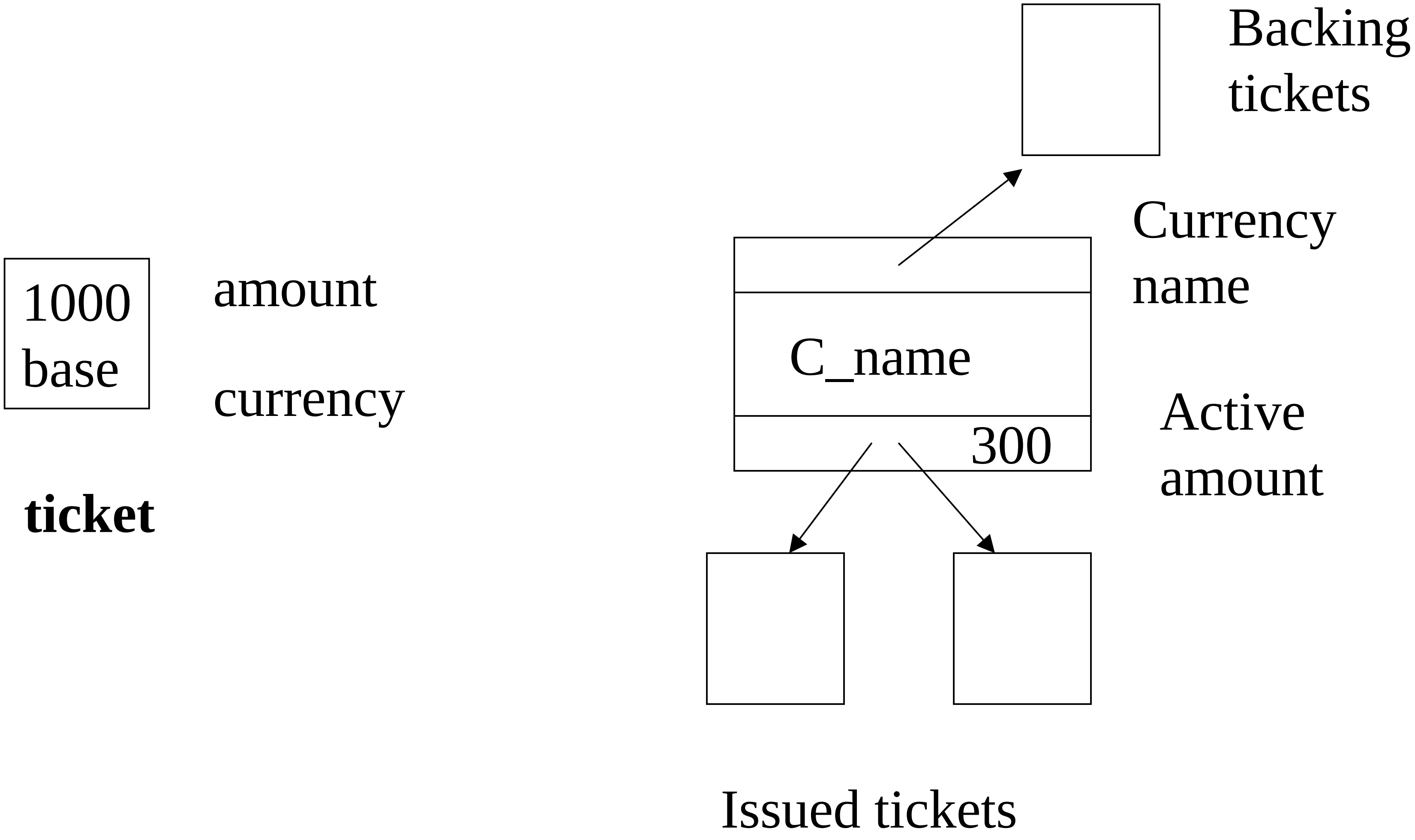
Summing: 10 12 17

Random(0, 19) = 15

Bells and Whistles

- Ticket transfers - objects that can be explicitly passed in messages
  - Can be used to solve priority inversions
- Ticket inflation
  - Create more - used among mutually trusting clients to dynamically adjust ticket allocations
- Currencies - “local” control, exchange rates
- Compensation tickets - to maintain share with I/O
  - use only ***f*** of quantum, ticket inflated by ***1/f*** in next

Kernel Objects



1000

base

amount

currency

ticket

Backing tickets

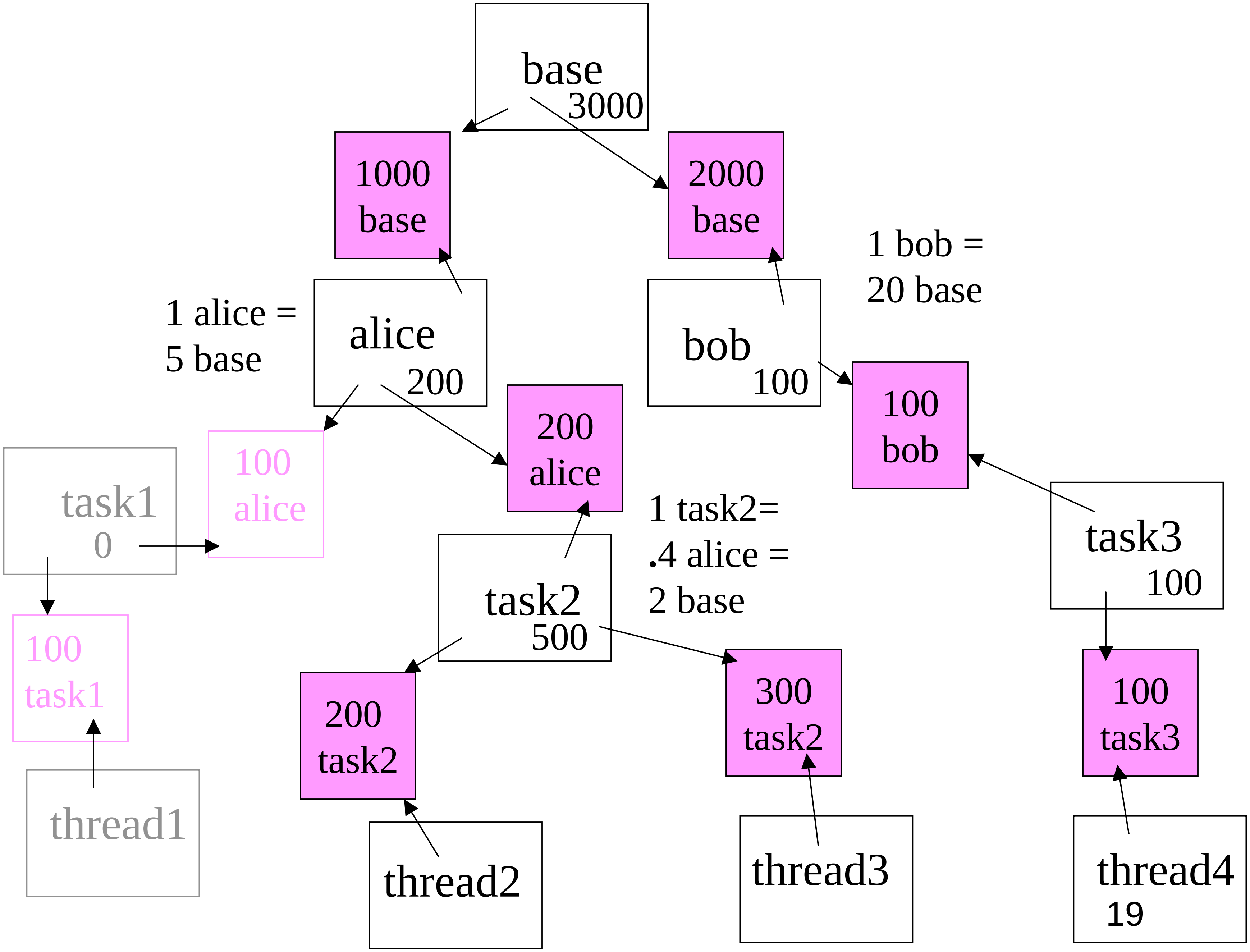
Currency name

C_name

300

Active amount

Issued tickets



base
3000

1000
base

2000
base

1 bob =
20 base

1 alice =
5 base

alice
200

bob
100

100
alice

200
alice

100
bob

task1
0

task2
500

1 task2=
.4 alice =
2 base

task3
100

100
task1

200
task2

300
task2

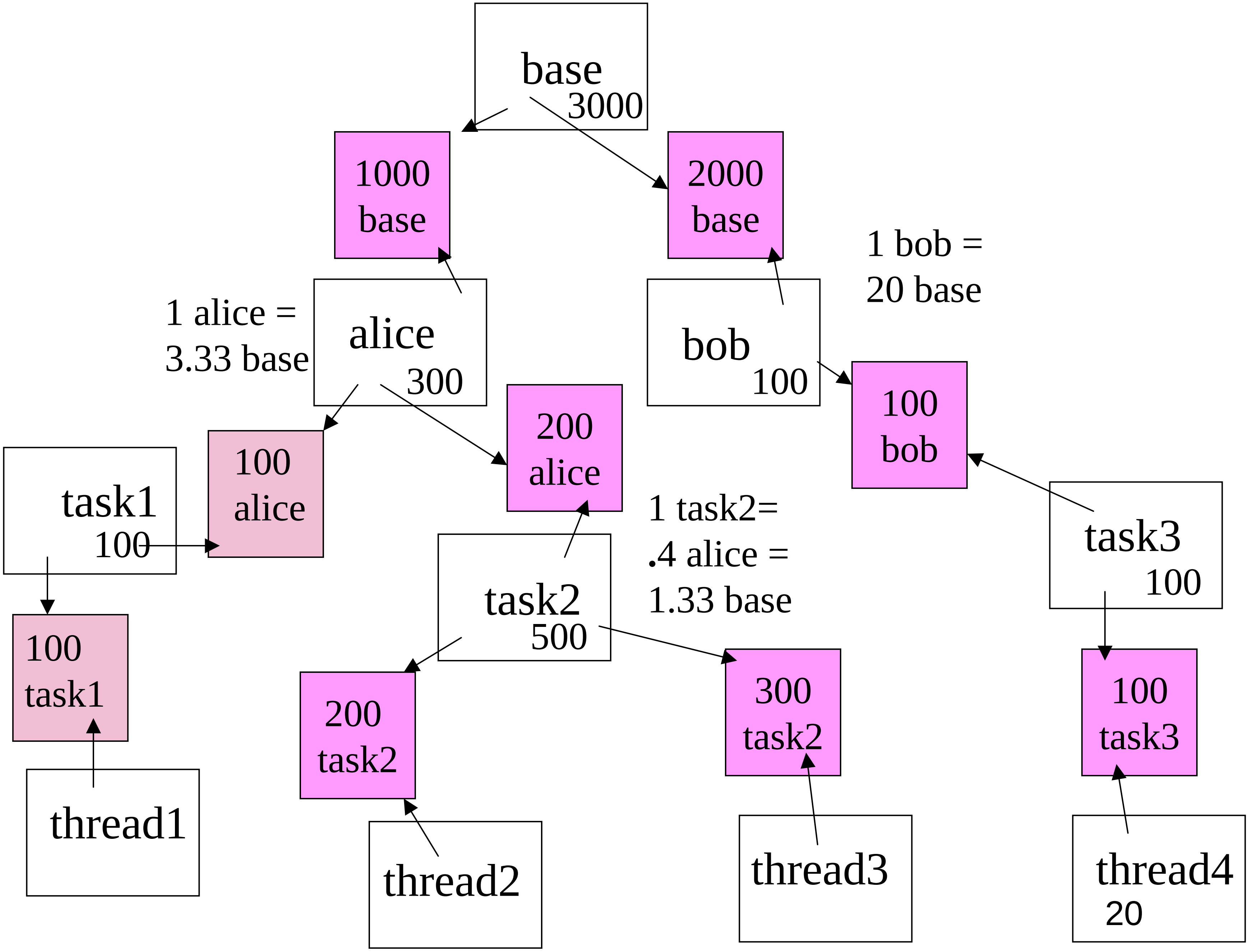
100
task3

thread1

thread2

thread3

thread4
19



base
3000

1000
base

2000
base

1 bob =
20 base

1 alice =
3.33 base

alice
300

bob
100

200
alice

100
bob

task1
100

100
alice

task2
500

1 task2=
.4 alice =
1.33 base

task3
100

100
task1

200
task2

300
task2

100
task3

thread1

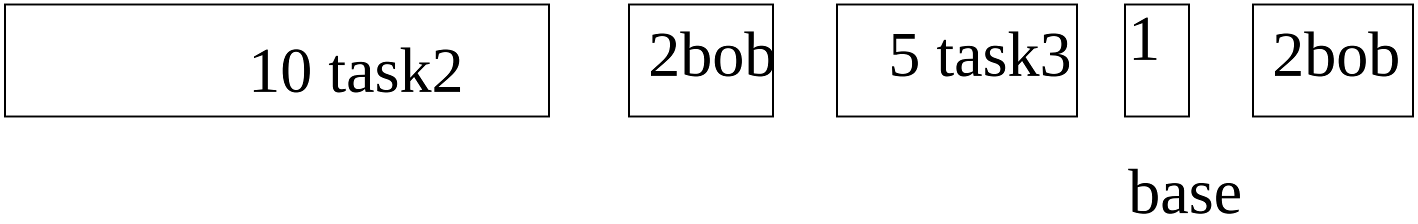
thread2

thread3

thread4
20

Example List-based Lottery

T = 3000 base



Random(0, 2999) = 1500

Compensation

- A holds 400 base, B holds 400 base
- A runs full 100msec quantum,
  B yields at 20msec
- B uses 1/5 allotted time
  Gets 400/(1/5) = 2000 base at each subsequent lottery for the rest of this quantum
  - a compensation ticket valued at 2000 - 400

Ticket Transfer

- Synchronous RPC between client and server
- create ticket in client’s currency and send to server to fund it’s currency
- on reply, the transfer ticket is destroyed

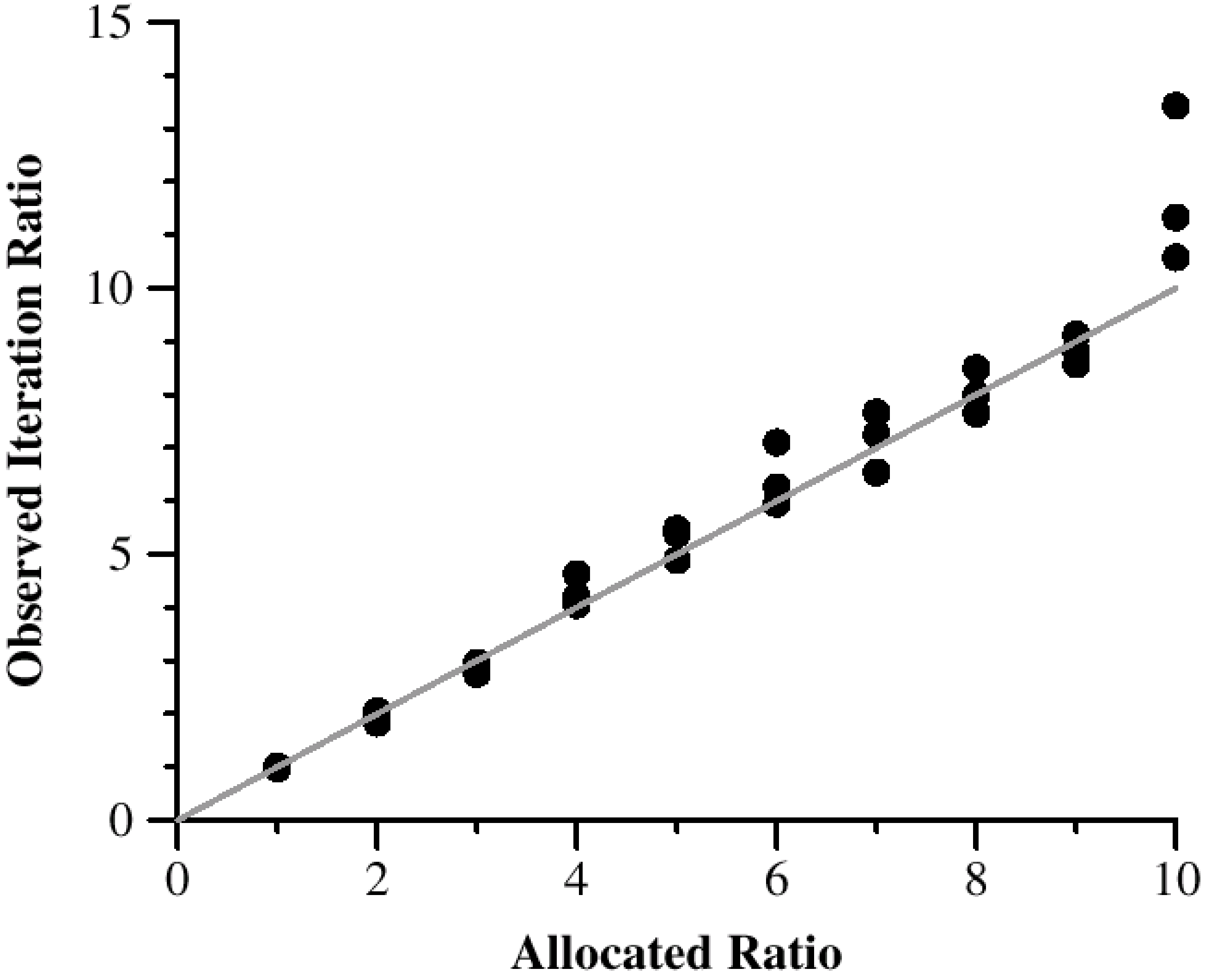
Control Scenarios

- Dynamic Control
  Conditionally and dynamically grant tickets
  Adaptability
- Resource abstraction barriers supported by currencies. Insulate tasks.

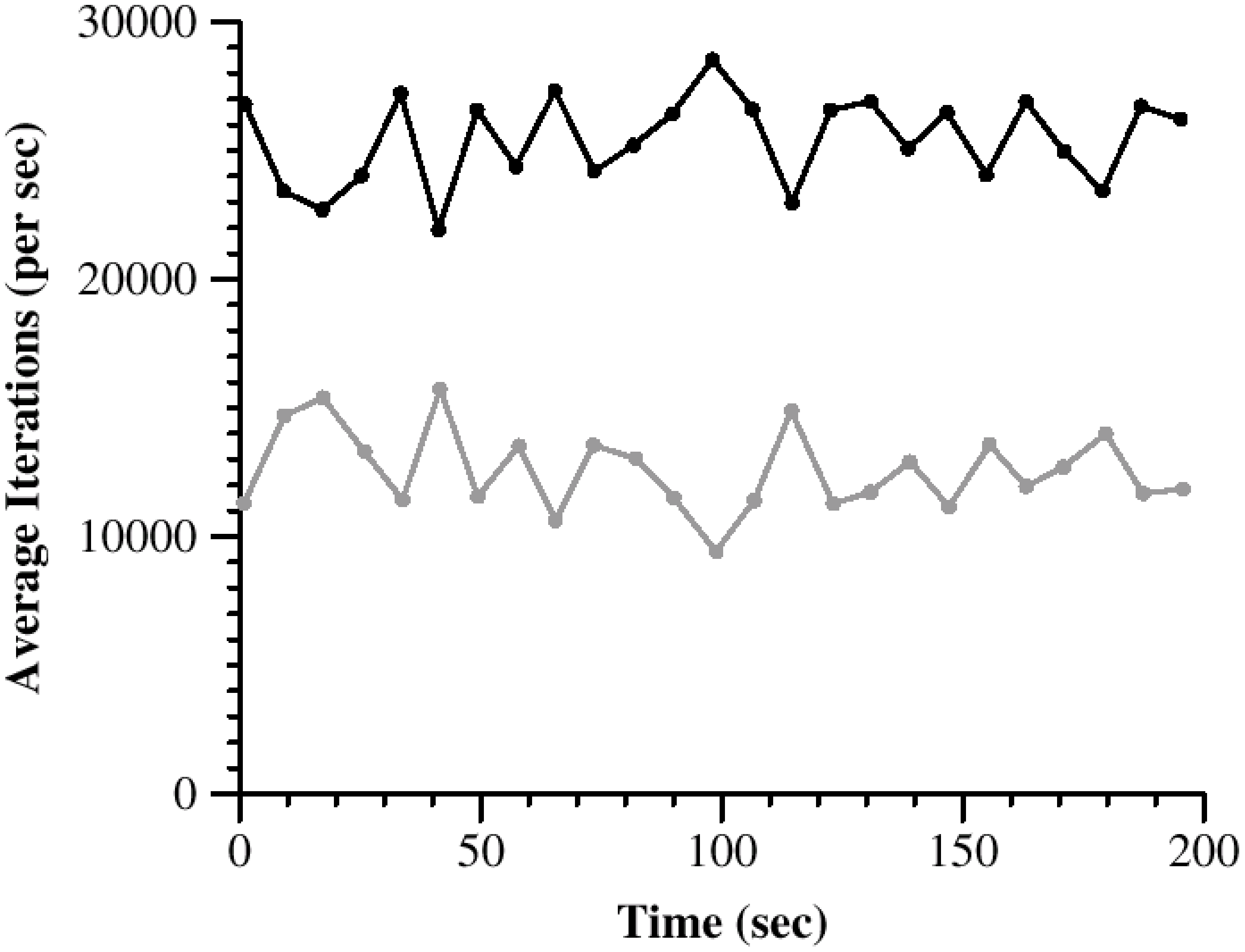
UI

- mktkt, rmtkt, mkcur, rmcur
- fund, unfund
- lstkt, lscur, fundx (shell)

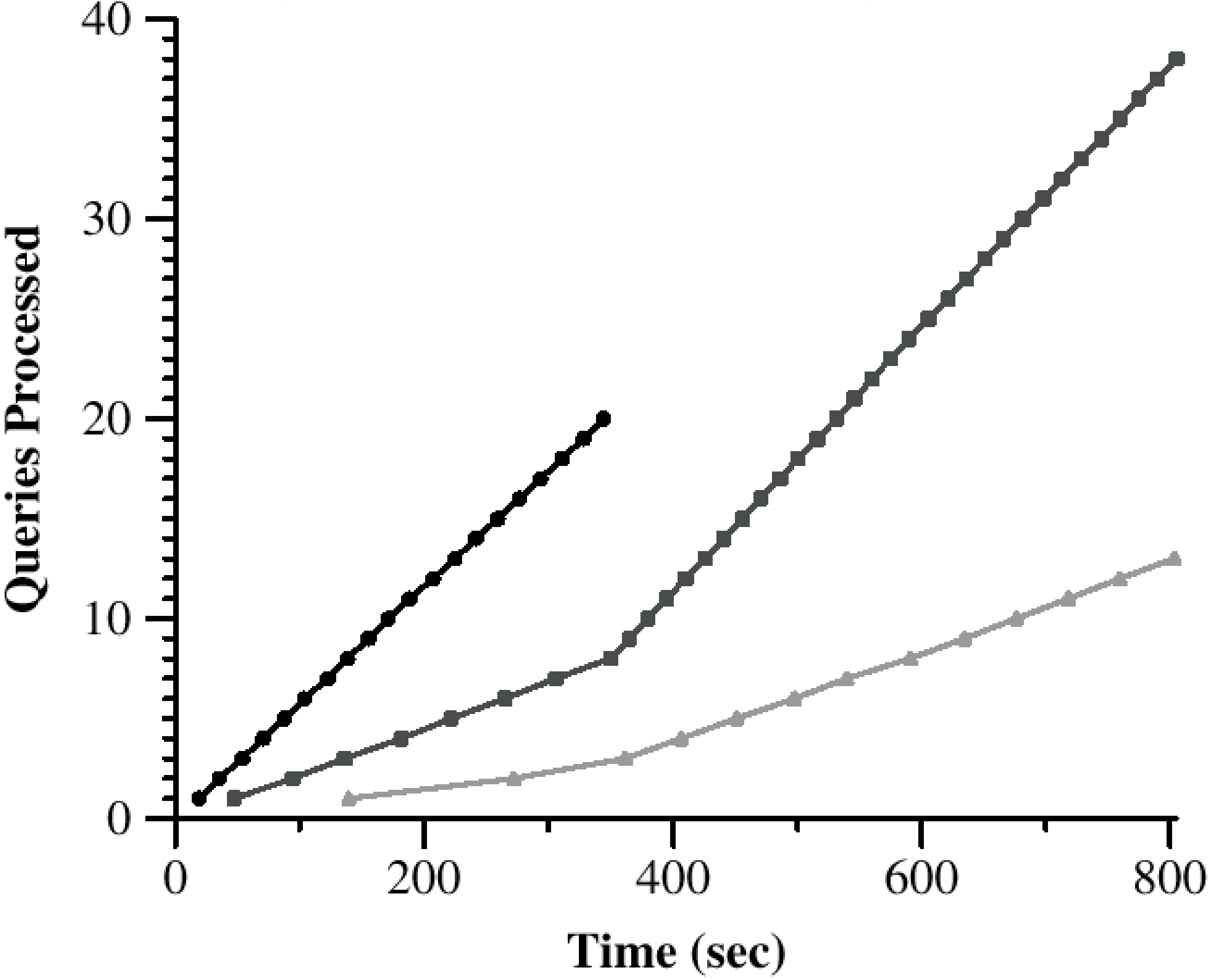
Relative Rate Accuracy



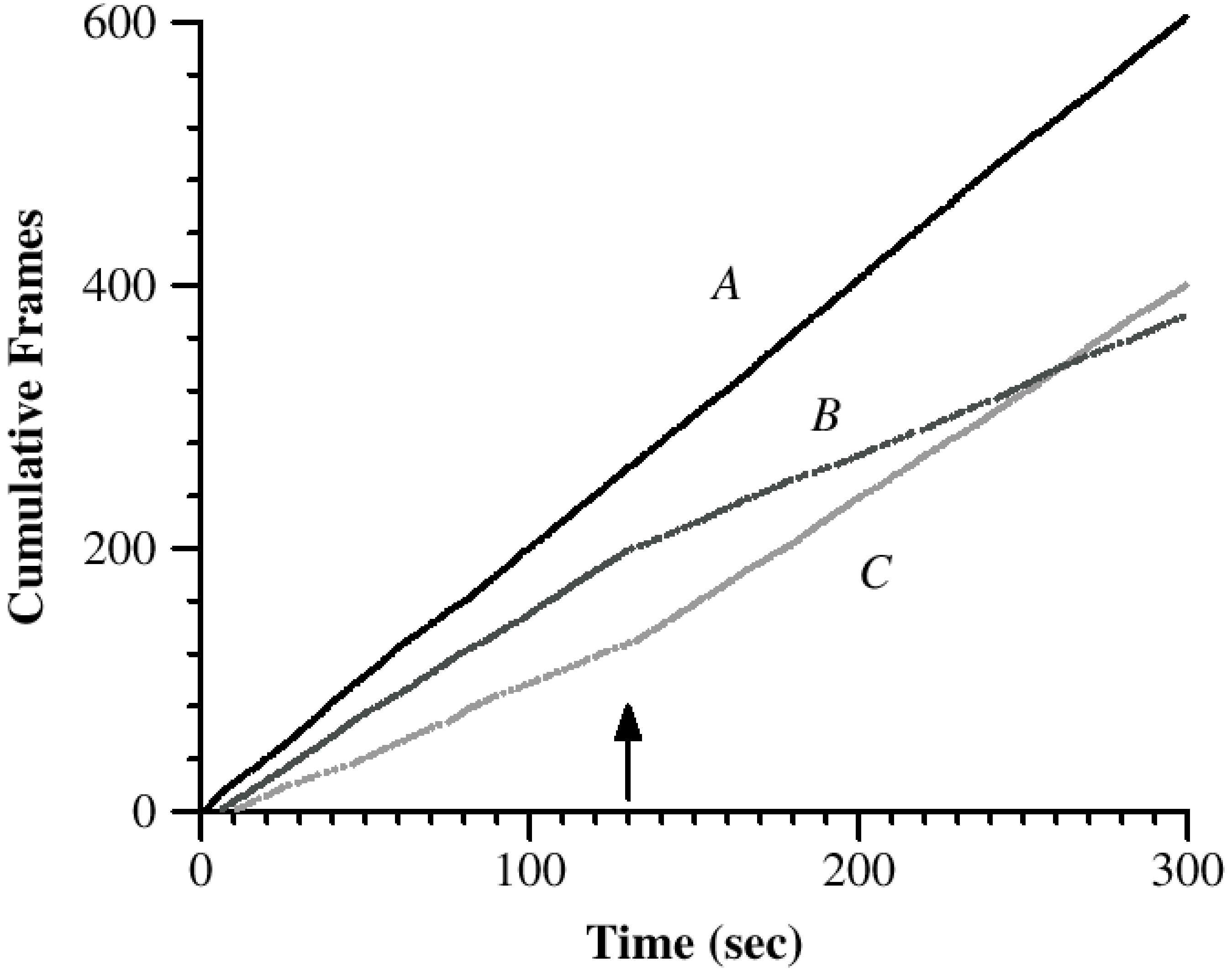
Fairness Over Time



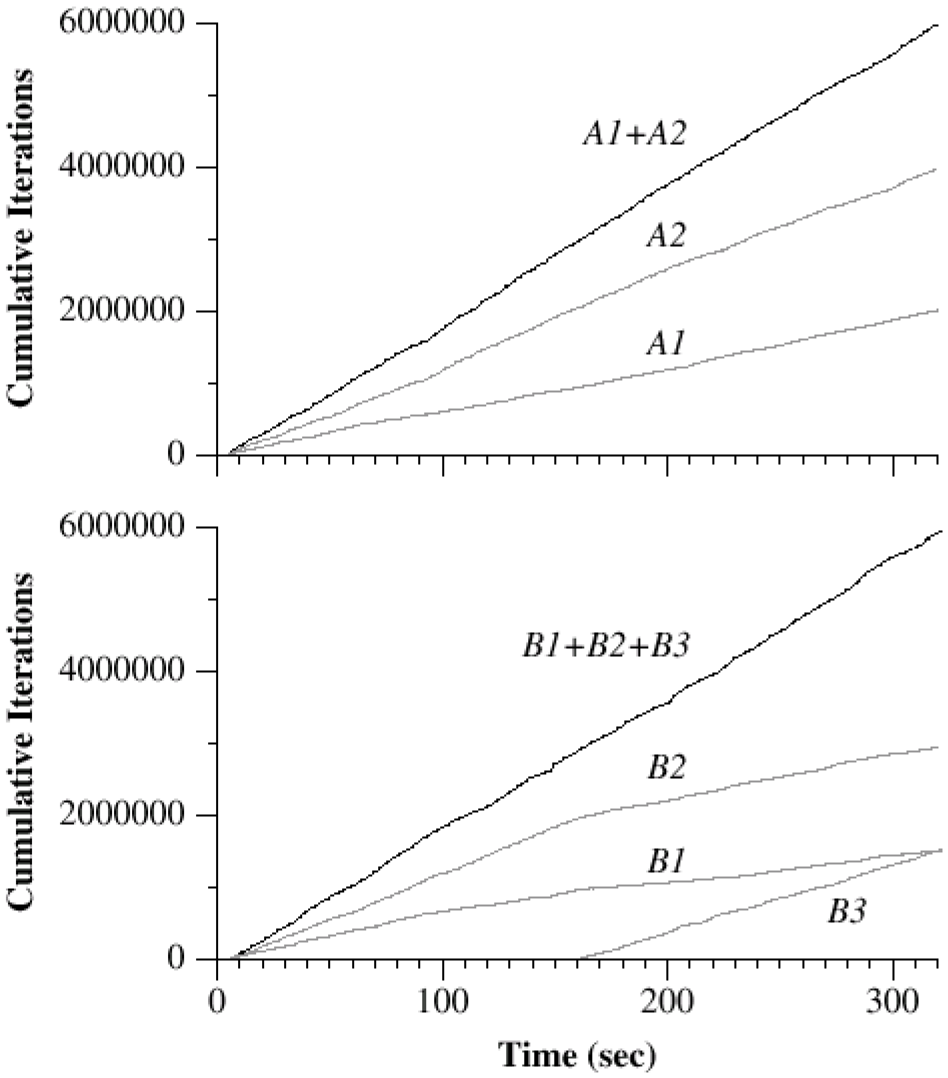
Client-Server
Query Processing Rates



Controlling Video Rates



Insulation

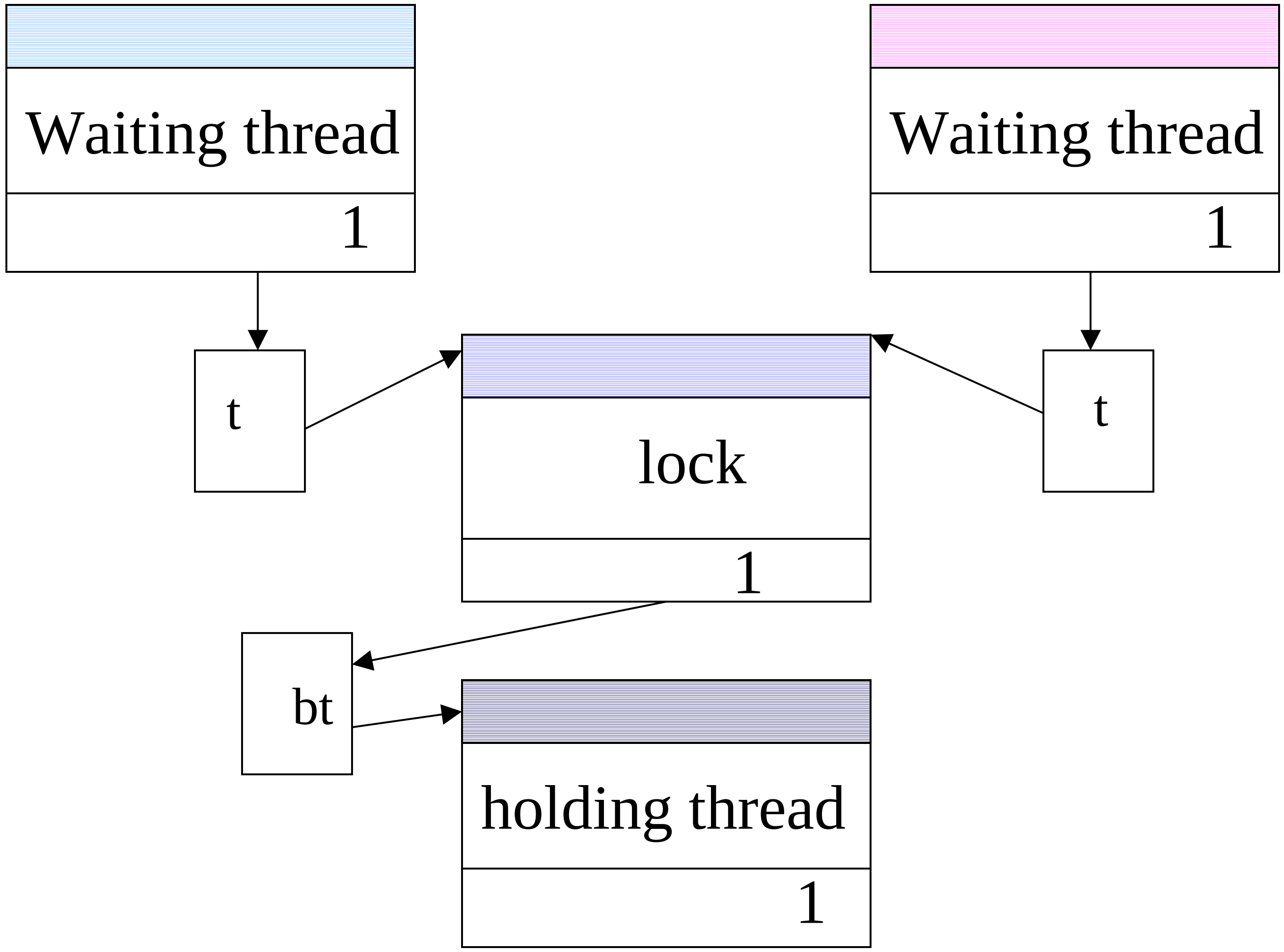


6000000
4000000
2000000
0
Cumulative Iterations
A1+A2
A2
A1
6000000
4000000
2000000
0
Cumulative Iterations
B1+B2+B3
B2
B1
B3
0
100
200
300
Time (sec)

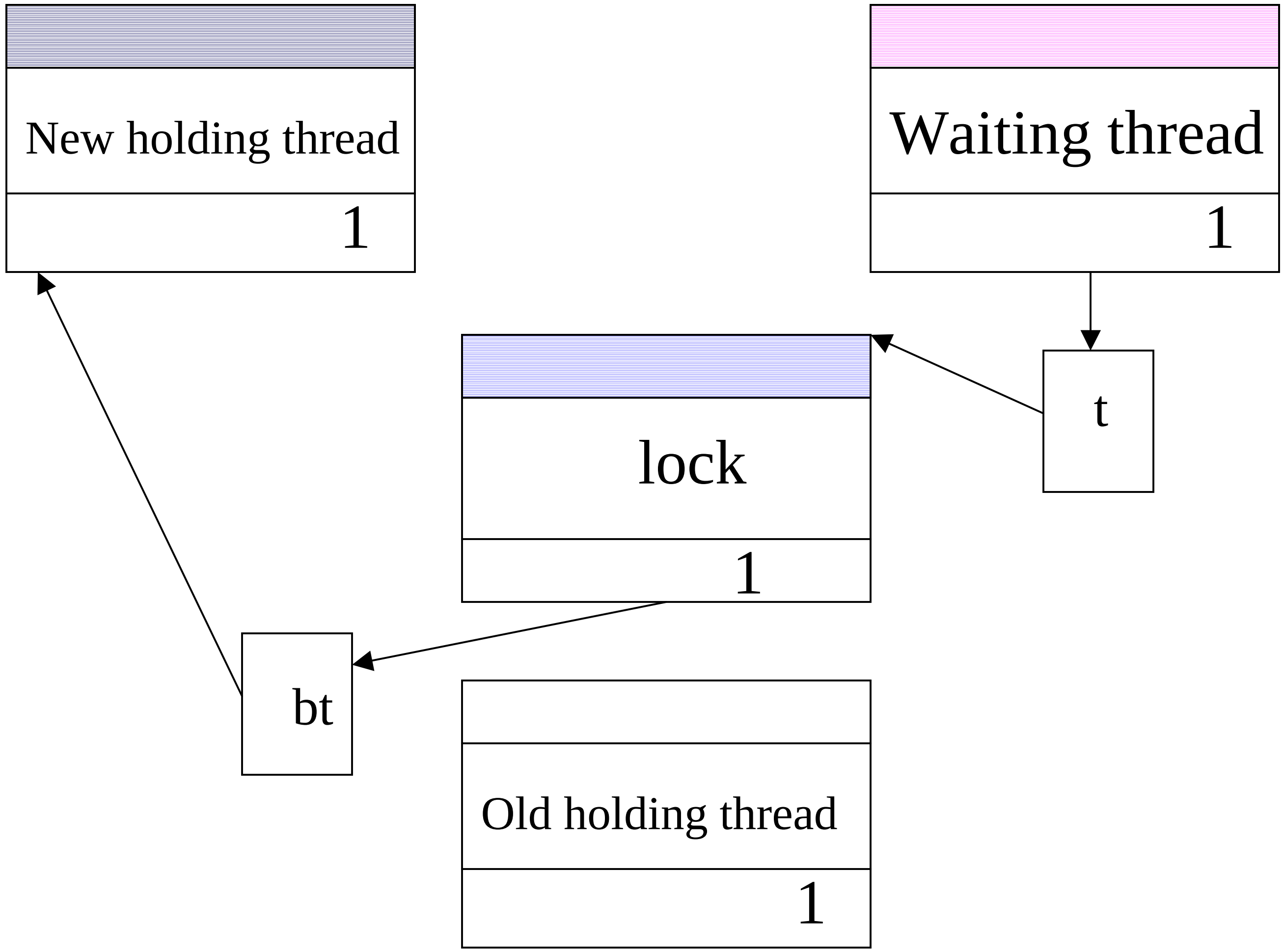
Other Kinds of Resources

- Claim: can be used for any resource where queuing is used
- Control relative waiting times for mutex locks.
  - Mutex currency funded out of currencies of waiting threads
  - Holder gets inheritance ticket in addition to its own funding, passed on to next holder (resulting from lottery) on release.
- Space sharing - inverse lottery, loser is victim (e.g. in page replacement decision, processor node preemption in MP partitioning)

Lock Funding



Lock Funding



Mutex Waiting Times