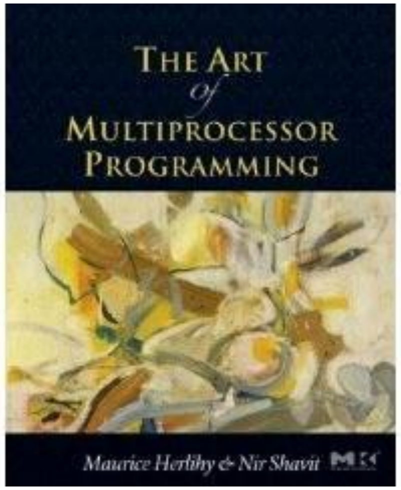
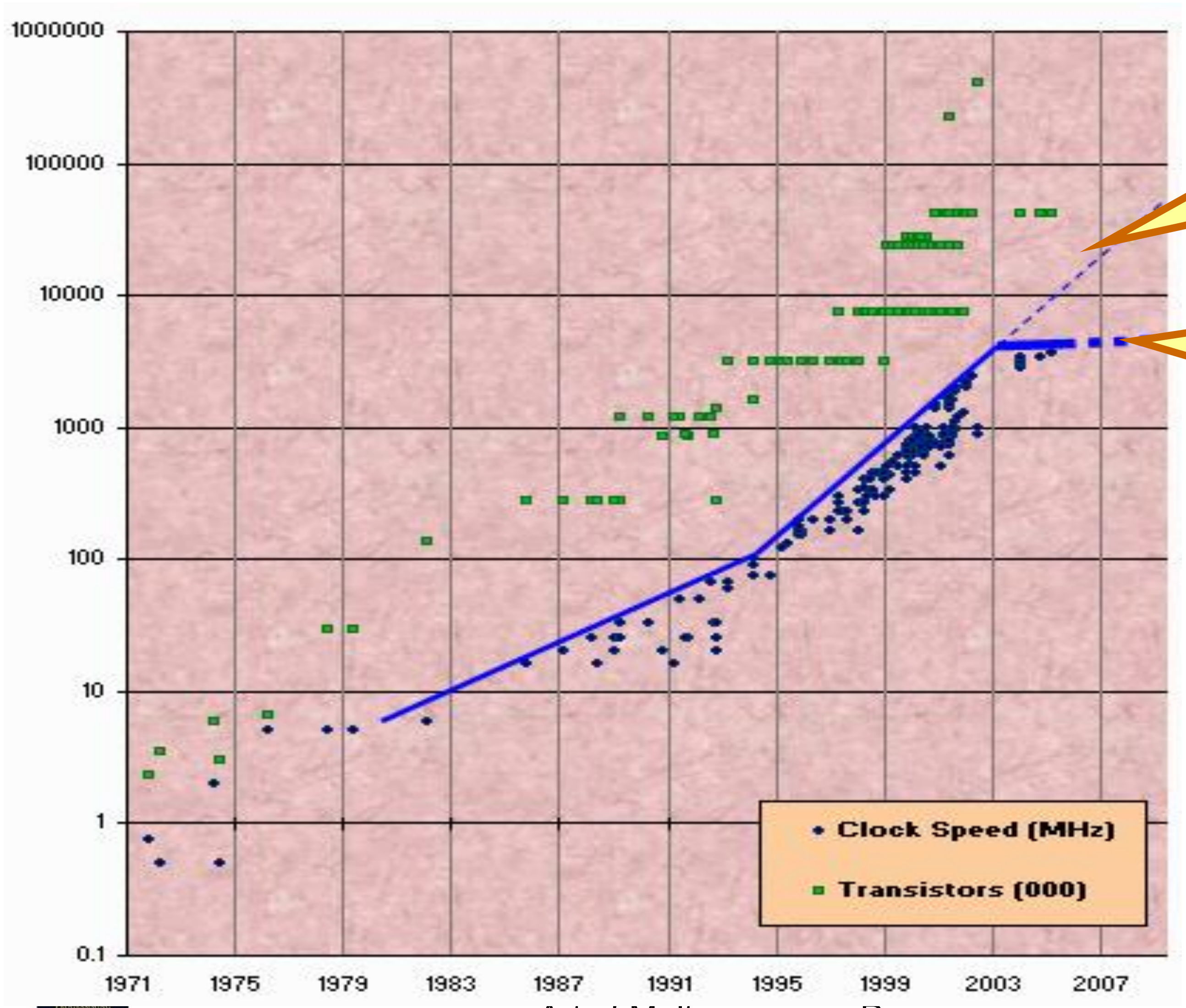
Introduction


THE ART of MULTIPROCESSOR PROGRAMMING
Maurice Herlihy & Nir Shavit
MK

Companion slides for
The Art of Multiprocessor Programming
by Maurice Herlihy & Nir Shavit

Moore's Law

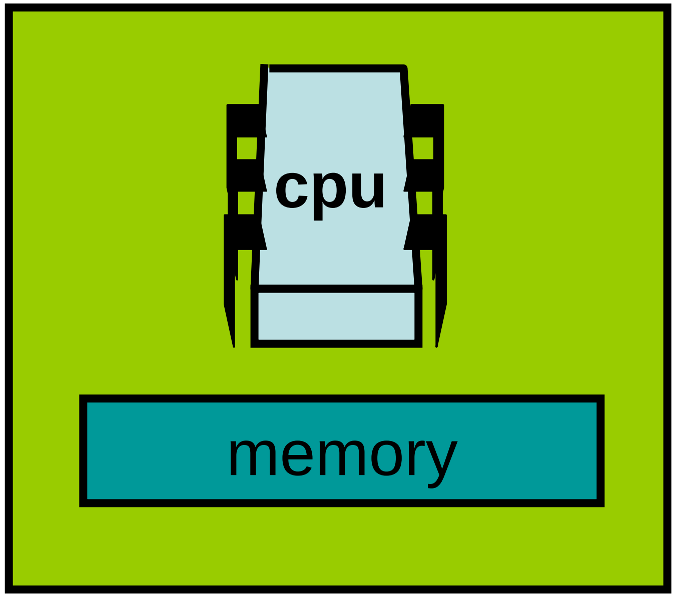


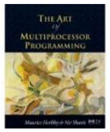
Transistor count still rising

Clock speed flattening sharply

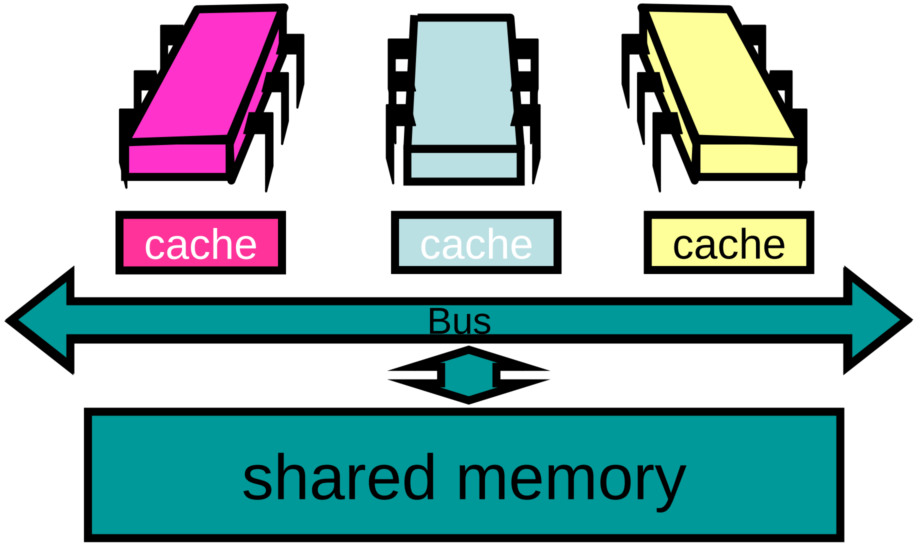
Moore's Law (in practice)

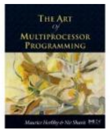
Nearly Extinct: the Uniprocesor





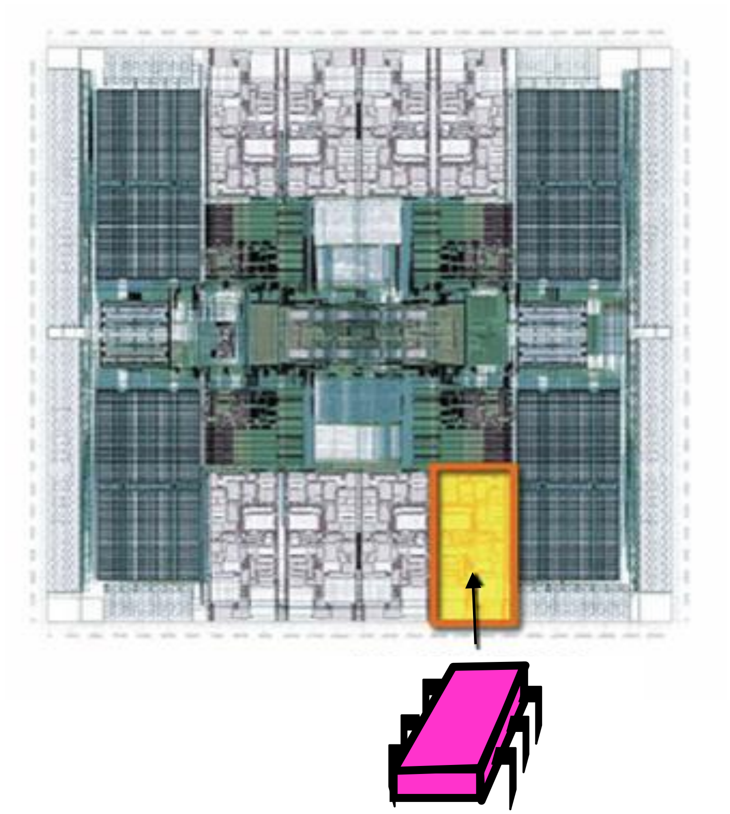
Endangered: The Shared Memory Multiprocessor (SMP)



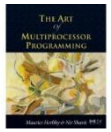


The New Boss: The Multicore Processor (CMP)

All on the same chip

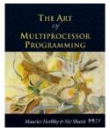


Sun T2000 Niagara



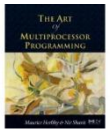
Why do we care?

- We want as much as possible to execute concurrently (in parallel)
- A larger sequential part implies reduced performance
- Amdahl’s law: this relation is not linear…



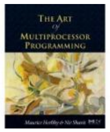
Amdahl's Law

$$\text{Speedup} = \frac{1\text{-thread execution time}}{n\text{-thread execution time}}$$



Amdahl's Law

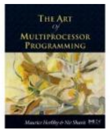
$$\textbf{Speedup=} \quad \frac{1}{1 \text{¡} \; p + \frac{p}{n}}$$



Amdahl's Law

$$\text{Speedup} = \frac{1}{1 - p + \boxed{\frac{p}{n}}}$$

Parallel fraction

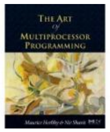


Amdahl’s Law

Parallel fraction

Speedup= $$\frac{1}{1 - p + \frac{p}{n}}$$

Number of threads



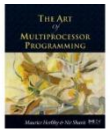
Amdahl's Law

Sequential fraction

Parallel fraction

$$\text{Speedup}=\frac{1}{\boxed{1-p}+\boxed{\frac{p}{\boxed{n}}}}$$

Number of threads

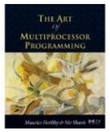


Amdahl’s Law in Practice

Bad synchronization ruins everything

Example

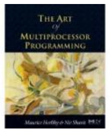
- Ten processors
- 60% concurrent, 40% sequential
- How close to 10-fold speedup?



Example

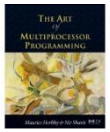
- Ten processors
- 60% concurrent, 40% sequential
- How close to 10-fold speedup?

Speedup = 2.17= $\frac{1}{1-0.6+\frac{0.6}{10}}$



Example

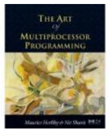
- Ten processors
- 80% concurrent, 20% sequential
- How close to 10-fold speedup?



Example

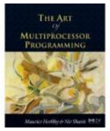
- Ten processors
- 80% concurrent, 20% sequential
- How close to 10-fold speedup?

$$\text{Speedup} = 3.57 = \frac{1}{1 - 0.8 + \frac{0.8}{10}}$$



Example

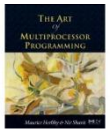
- Ten processors
- 90% concurrent, 10% sequential
- How close to 10-fold speedup?



Example

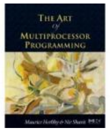
- Ten processors
- 90% concurrent, 10% sequential
- How close to 10-fold speedup?

$$\text{Speedup} = 5.26 = \frac{1}{1 - 0.9 + \frac{0.9}{10}}$$



Example

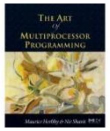
- Ten processors
- 99% concurrent, 01% sequential
- How close to 10-fold speedup?



Example

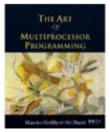
- Ten processors
- 99% concurrent, 01% sequential
- How close to 10-fold speedup?

Speedup = 9.17= $\dfrac{1}{1-0.99+\dfrac{0.99}{10}}$



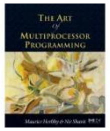
Concurrent Objects

- What is a concurrent object?
  - How do we **describe** one?
  - How do we **implement** one?
  - How do we **tell if we're right**?



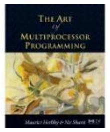
Sequential Objects

- Each object has a ***state***
  - Usually given by a set of ***fields***
  - Queue example: sequence of items
- Each object has a set of ***methods***
  - Only way to manipulate state
  - Queue example: **enq** and **deq** methods



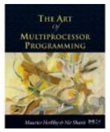
Sequential Specifications

- If (precondition)
  - the object is in such-and-such a state
  - before you call the method,
- Then (postcondition)
  - the method will return a particular value
  - or throw a particular exception.
- and (postcondition, con't)
  - the object will be in some other state
  - when the method returns,



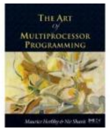
Pre and PostConditions for Dequeue

- Precondition:
  - Queue is non-empty
- Postcondition:
  - Returns first item in queue
- Postcondition:
  - Removes first item in queue



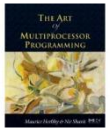
Pre and PostConditions for Dequeue

- Precondition:
  - Queue is empty
- Postcondition:
  - Throws Empty exception
- Postcondition:
  - Queue state unchanged



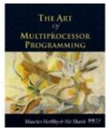
Sequential Specifications

- Interactions among methods captured by side-effects on object state
  - State meaningful between method calls
- Documentation size linear in number of methods
  - Each method described in isolation
- Can add new methods
  - Without changing descriptions of old methods

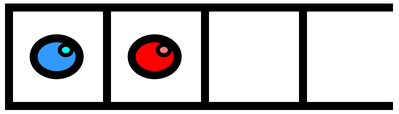


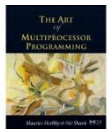
What About Concurrent Specifications ?

- Methods?
- Documentation?
- Adding new methods?

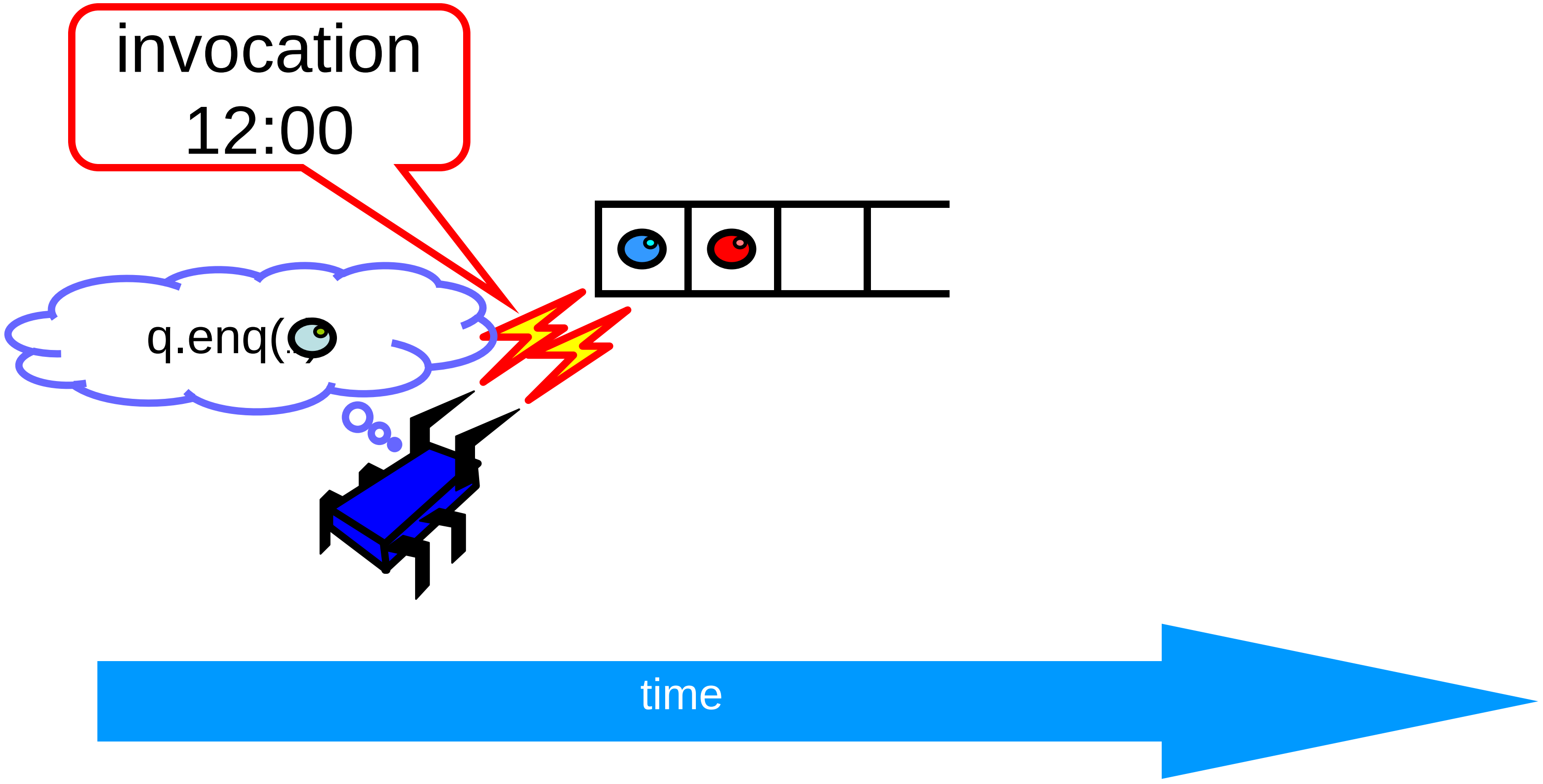


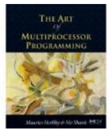
Methods Take Time



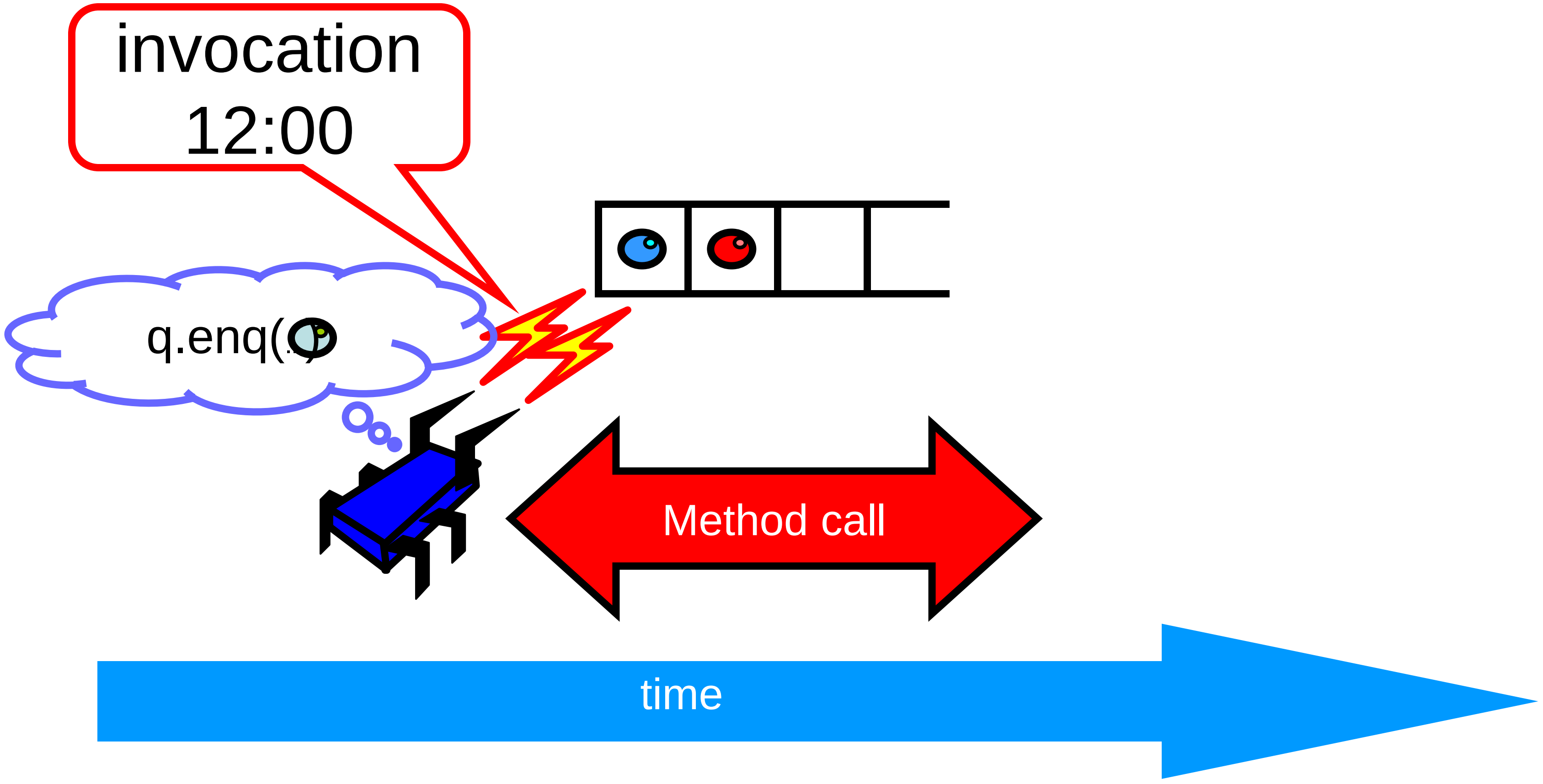


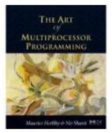
Methods Take Time



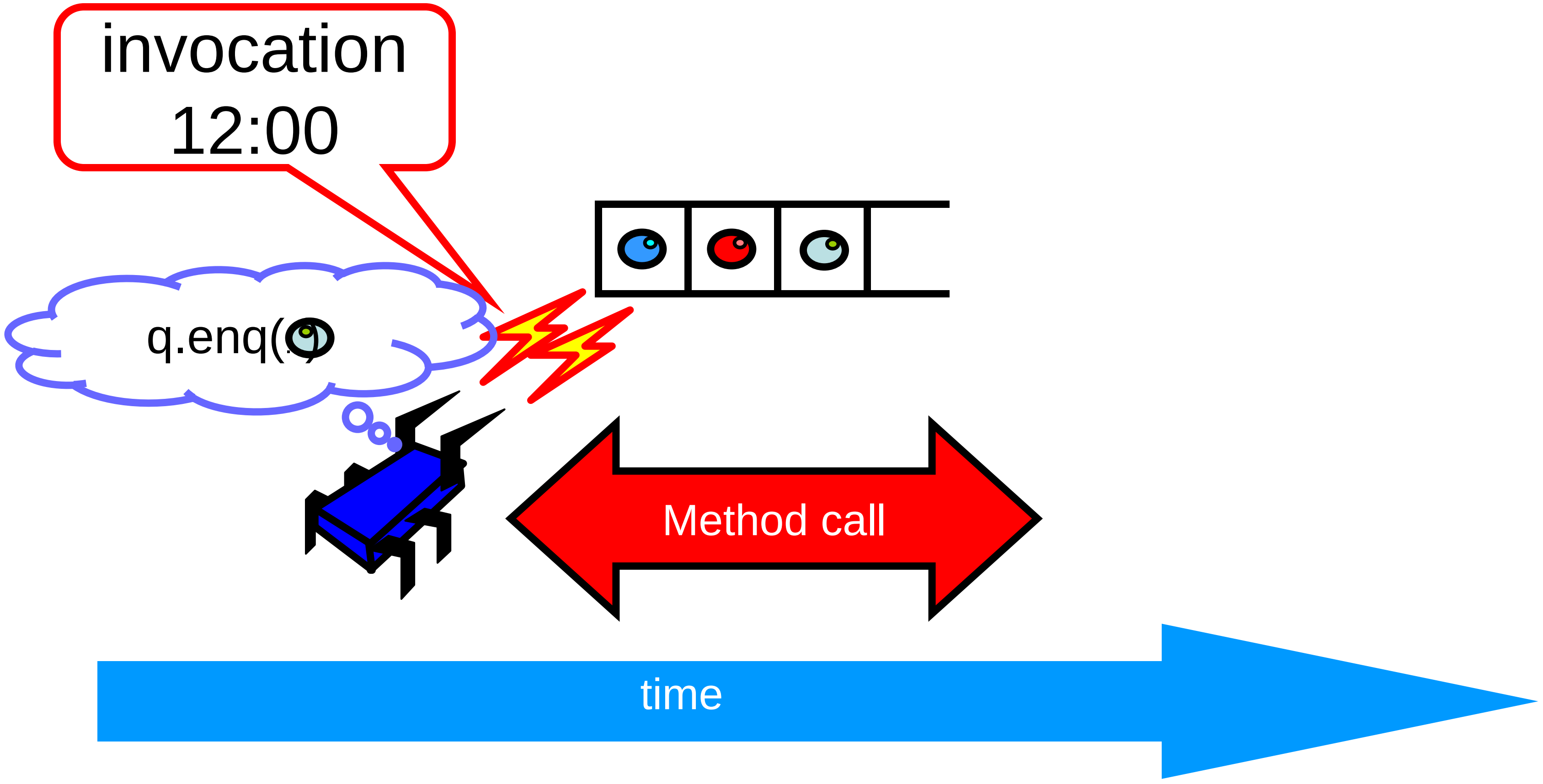


Methods Take Time

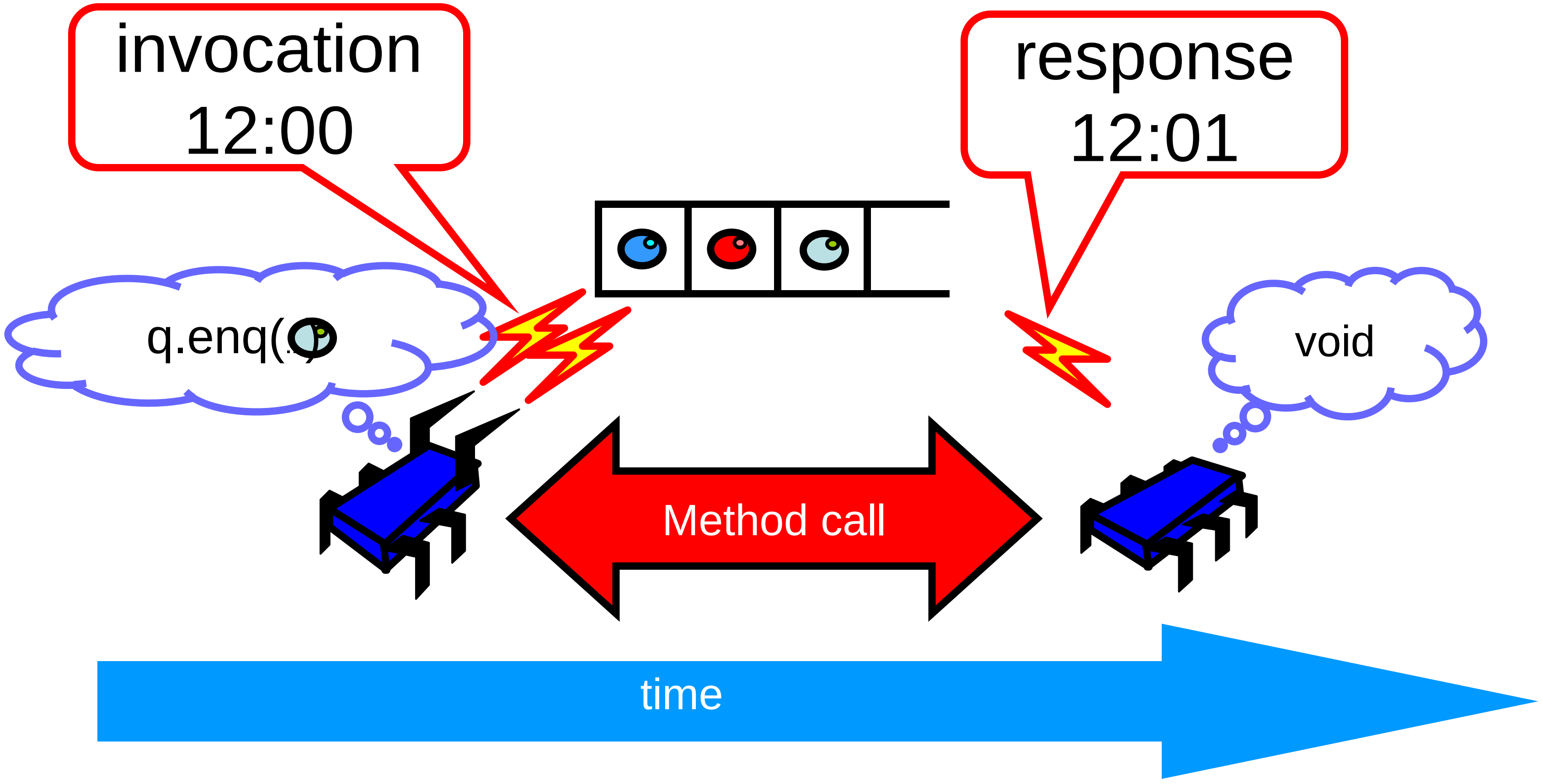


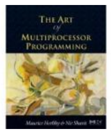


Methods Take Time

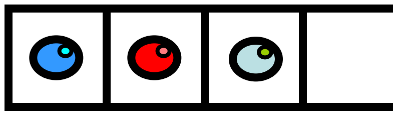


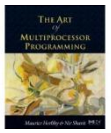
Methods Take Time



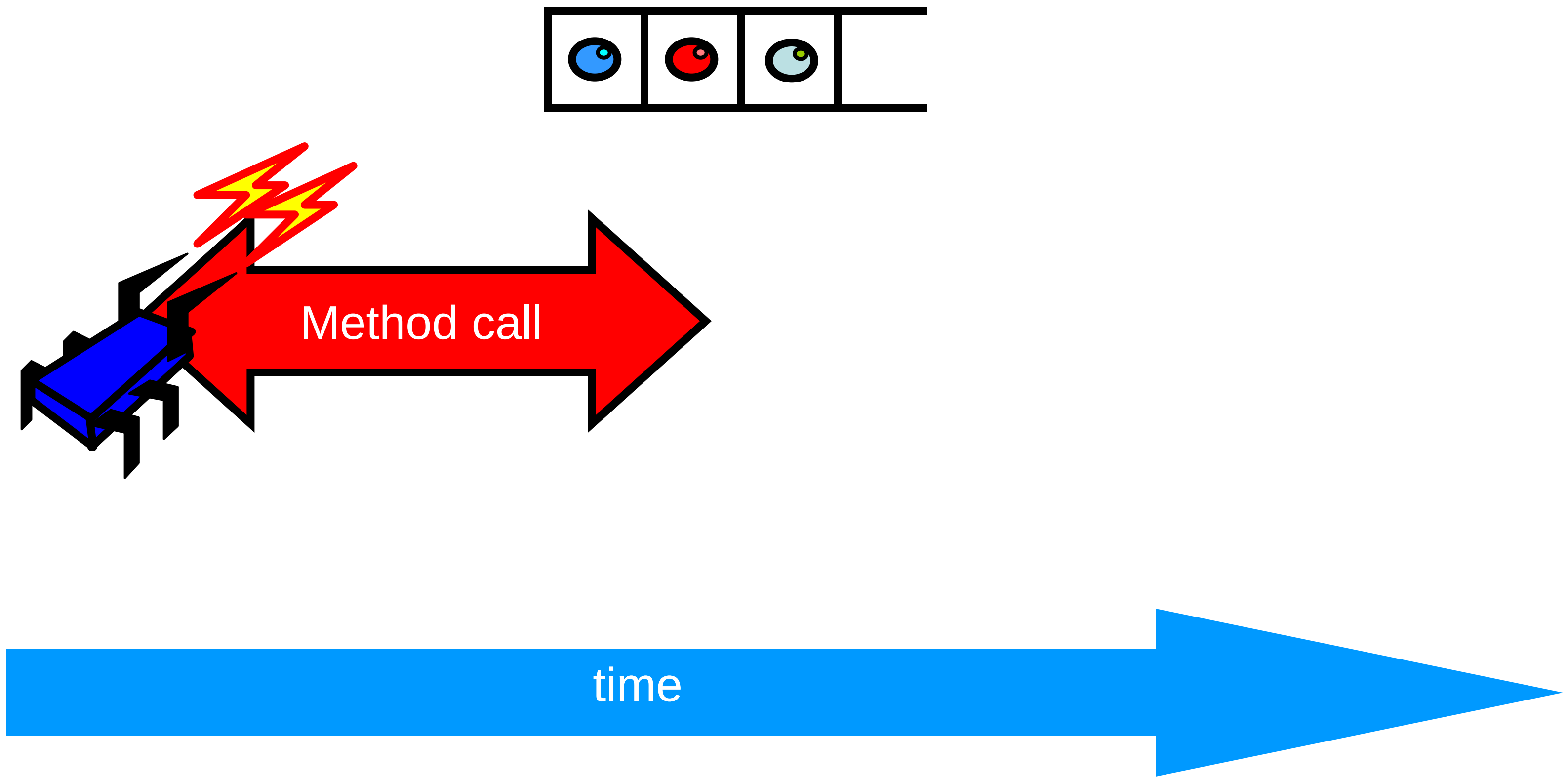


Concurrent Methods Take Overlapping Time

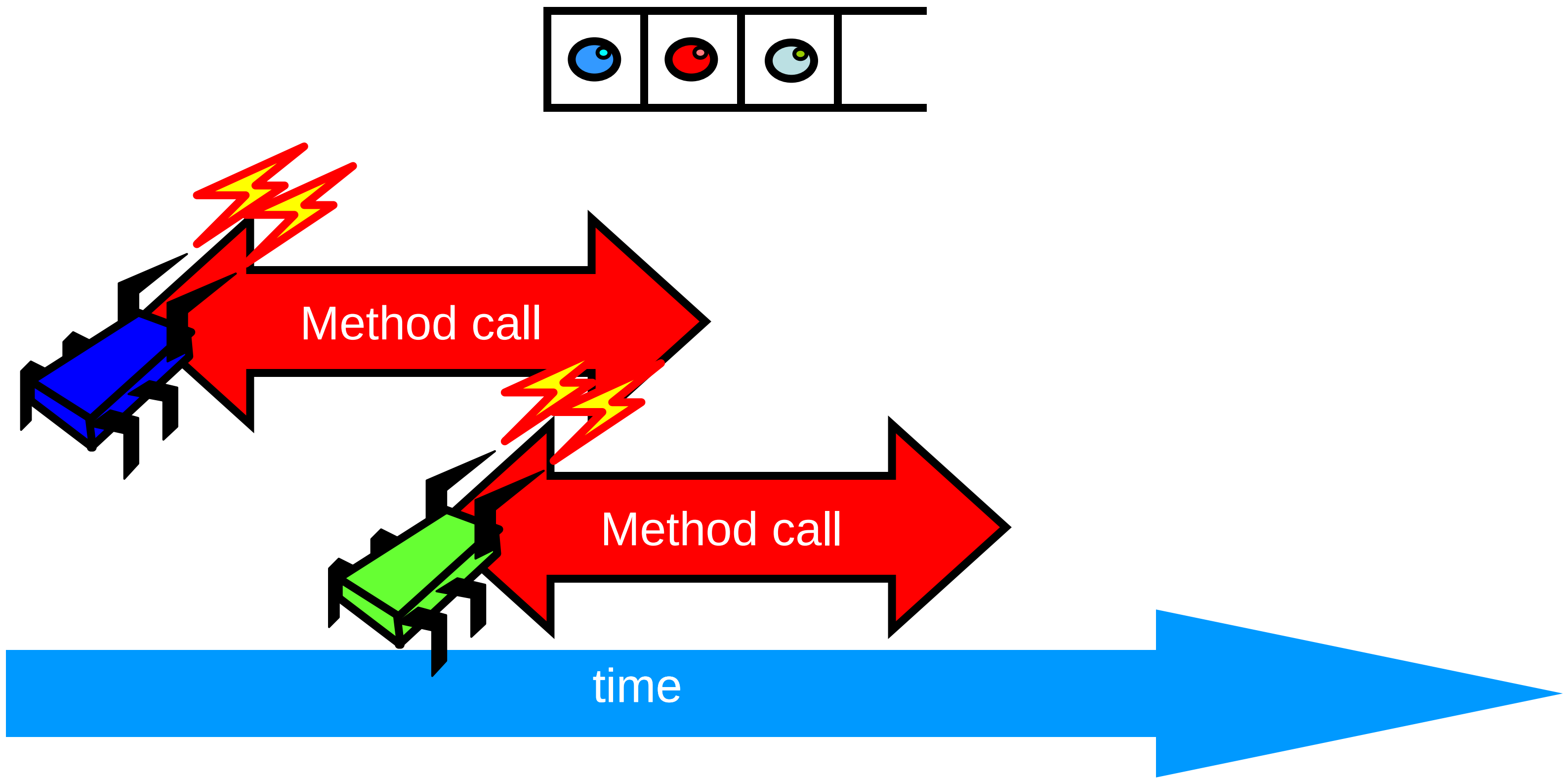


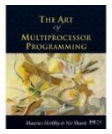


Concurrent Methods Take Overlapping Time

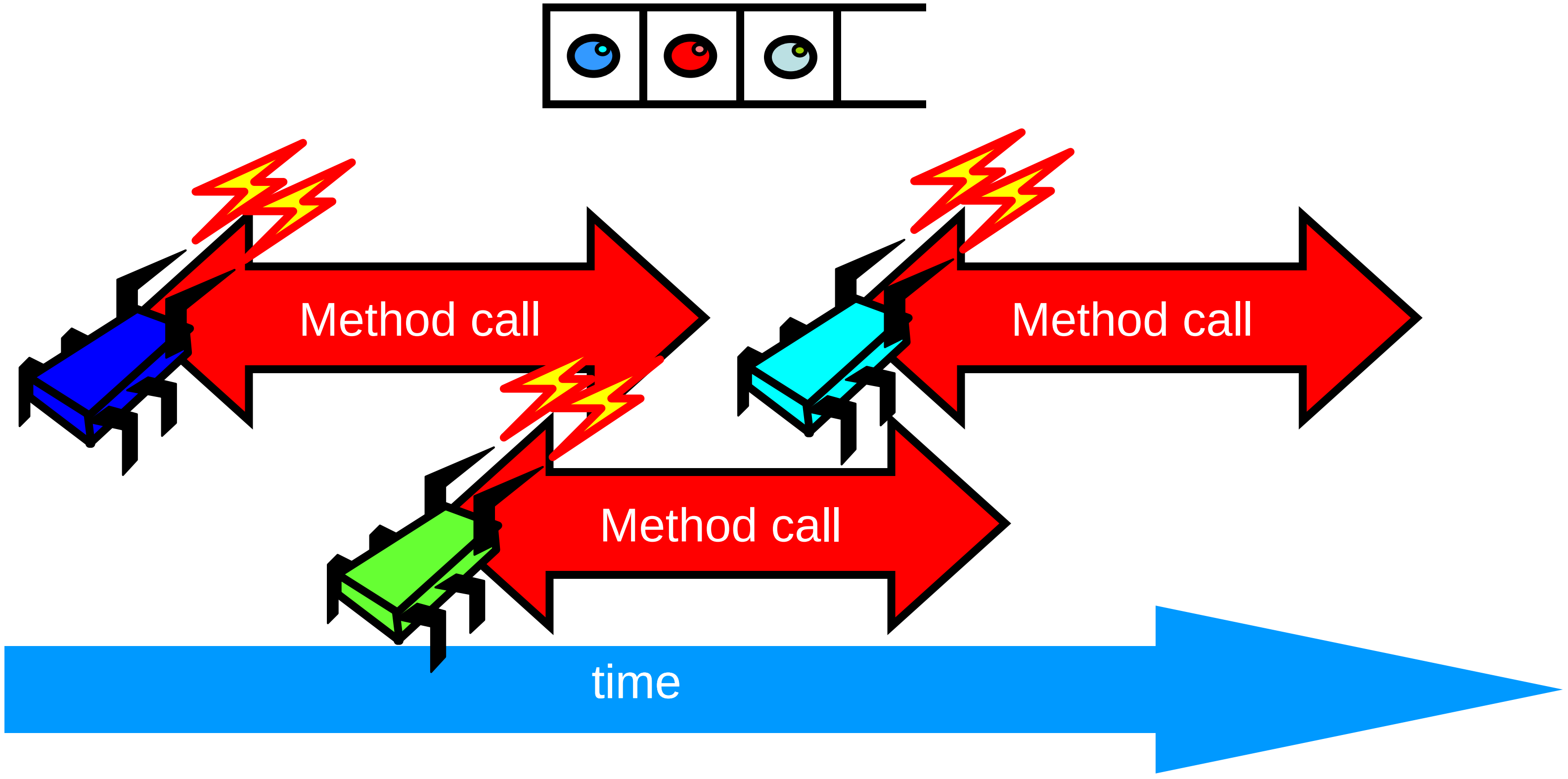


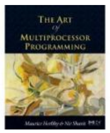
Concurrent Methods Take Overlapping Time





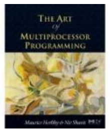
Concurrent Methods Take Overlapping Time



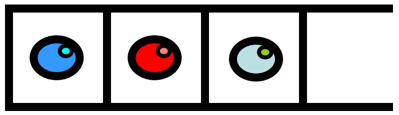


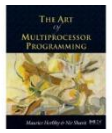
Linearizability

- Each method should
  - "take effect"
  - Instantaneously
  - Between invocation and response events
- Object is correct if this "sequential" behavior is correct
- Any such concurrent object is
  - **Linearizable™**

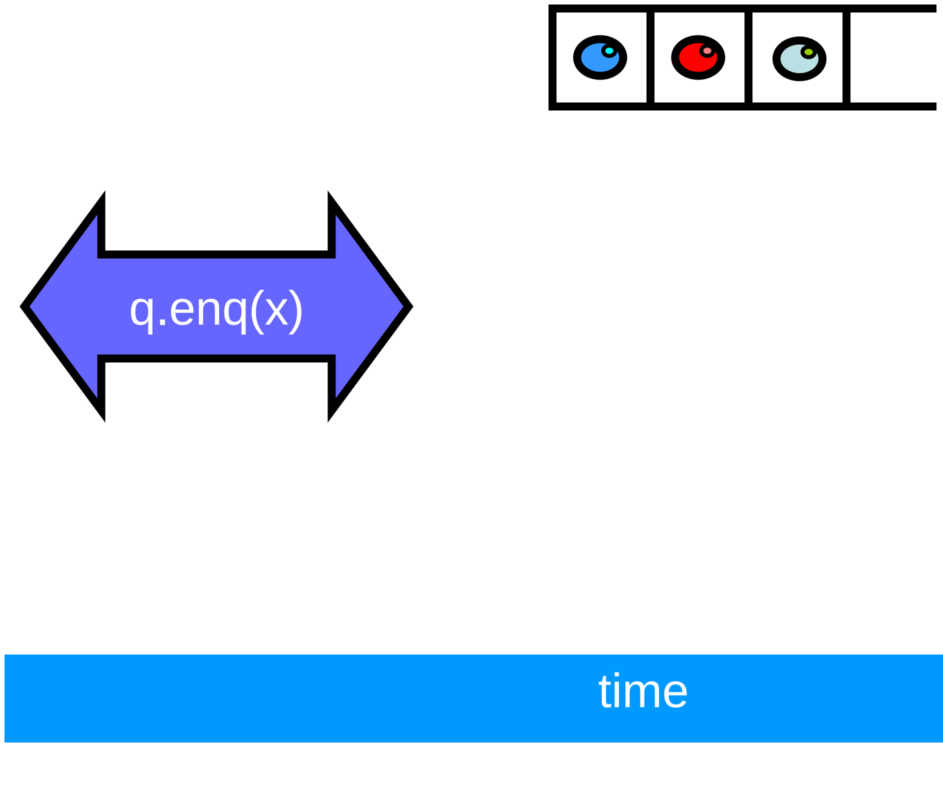


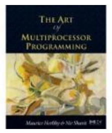
Example



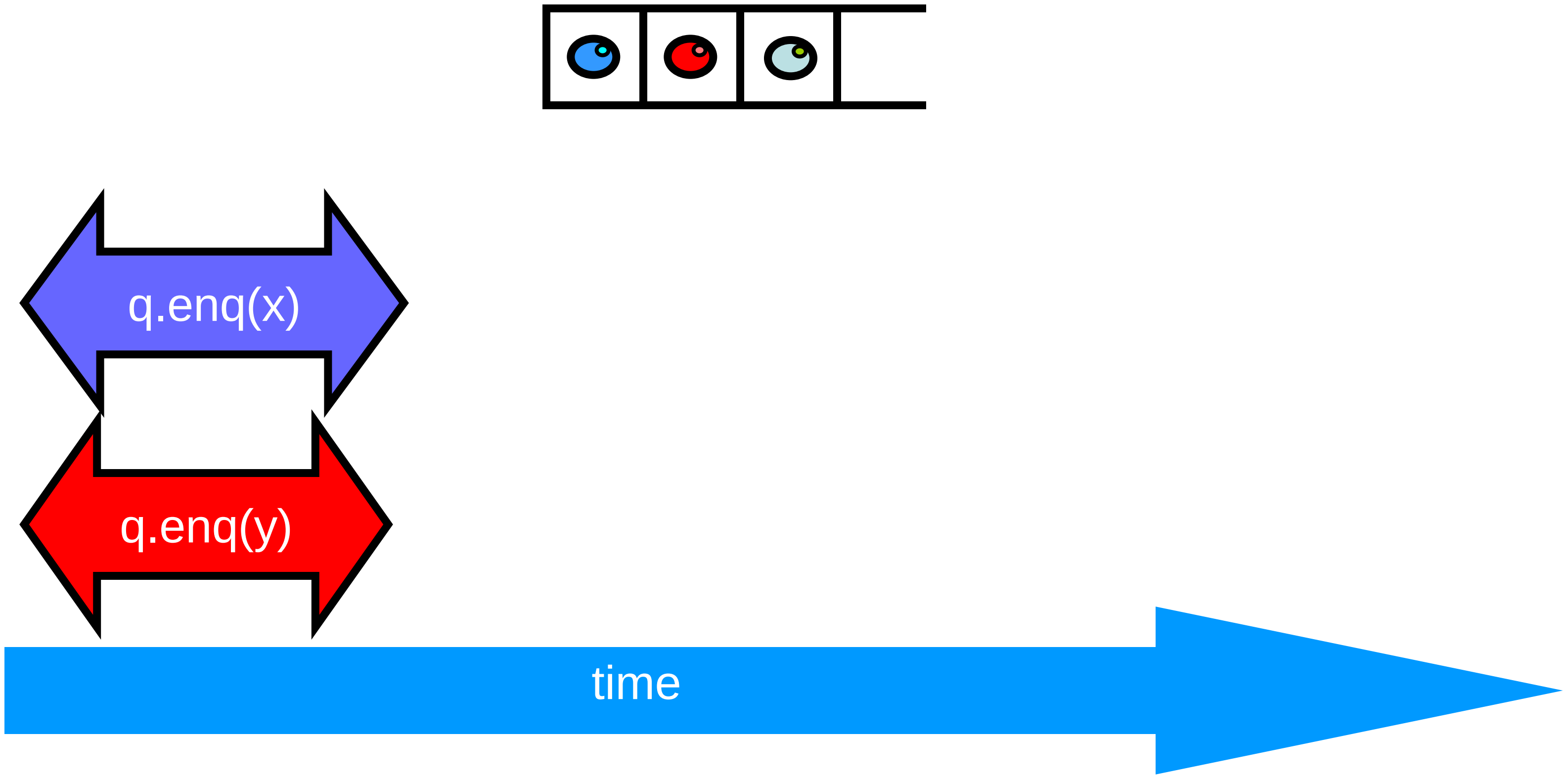


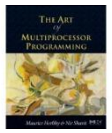
Example



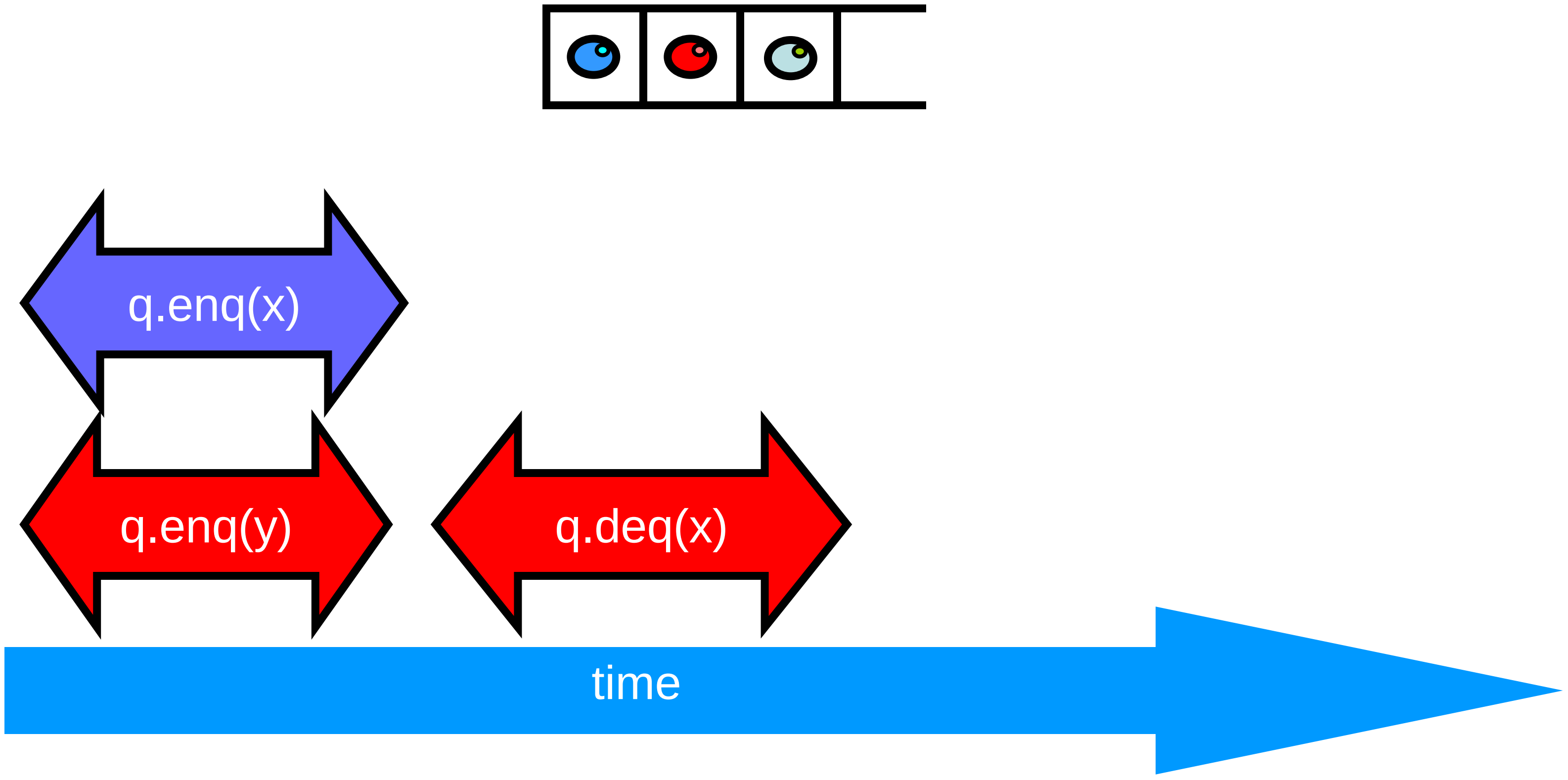


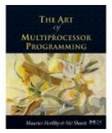
Example



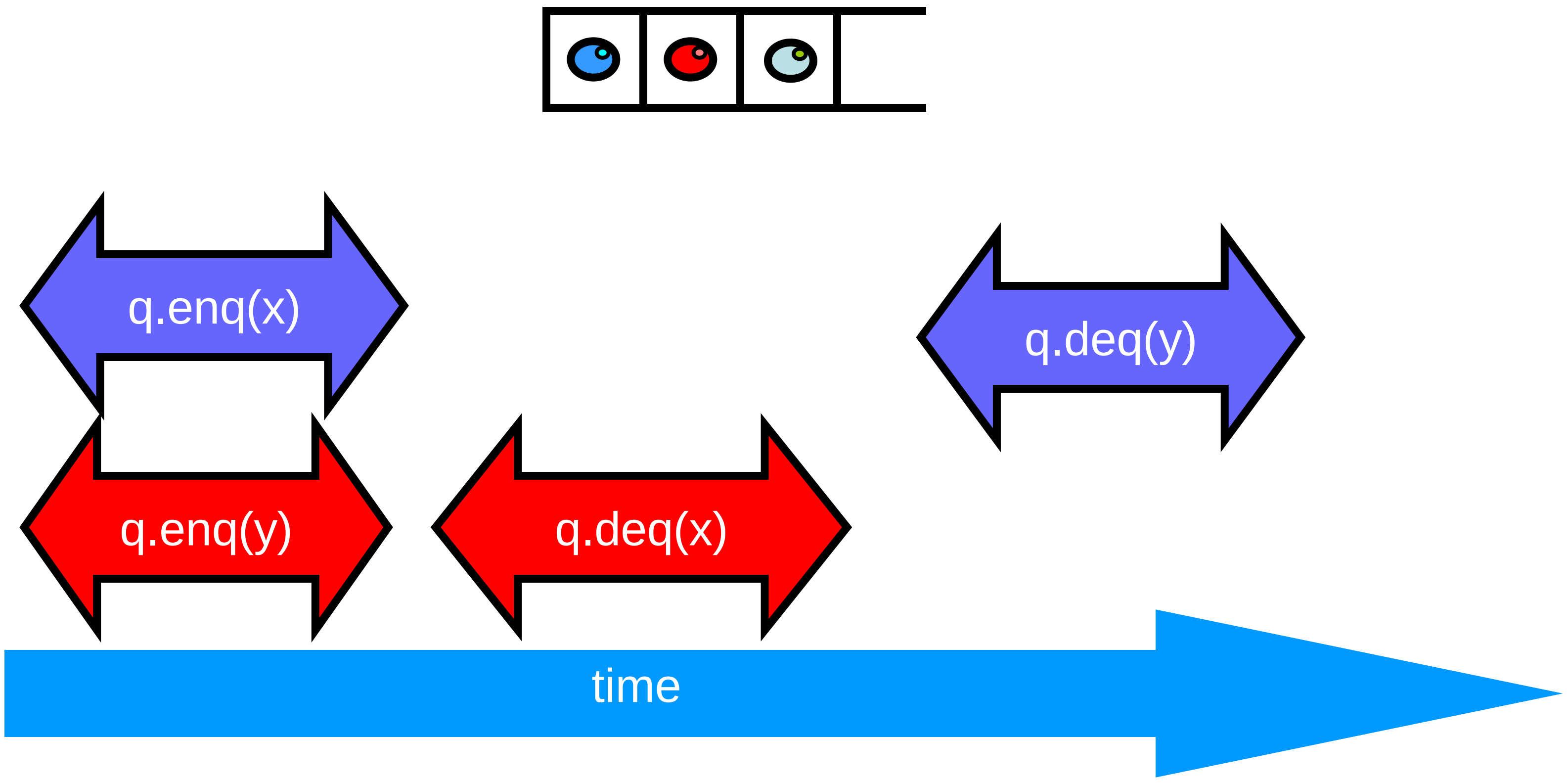


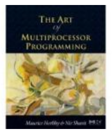
Example



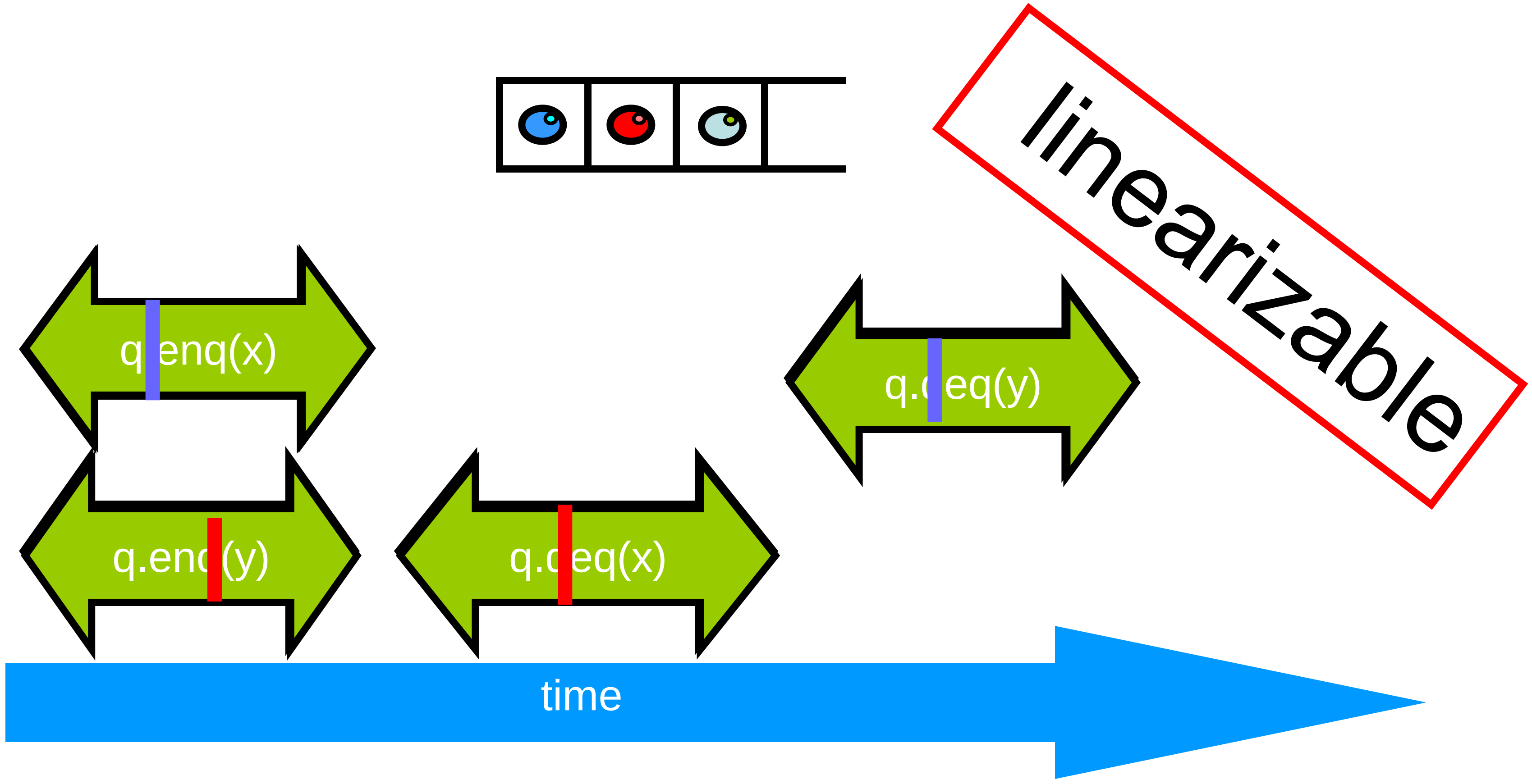


Example





Example


linearizable
q.enq(x)
q.deq(y)
q.enq(y)
q.deq(x)
time