

# Tiptoe: A Compositional Real-Time Operating System

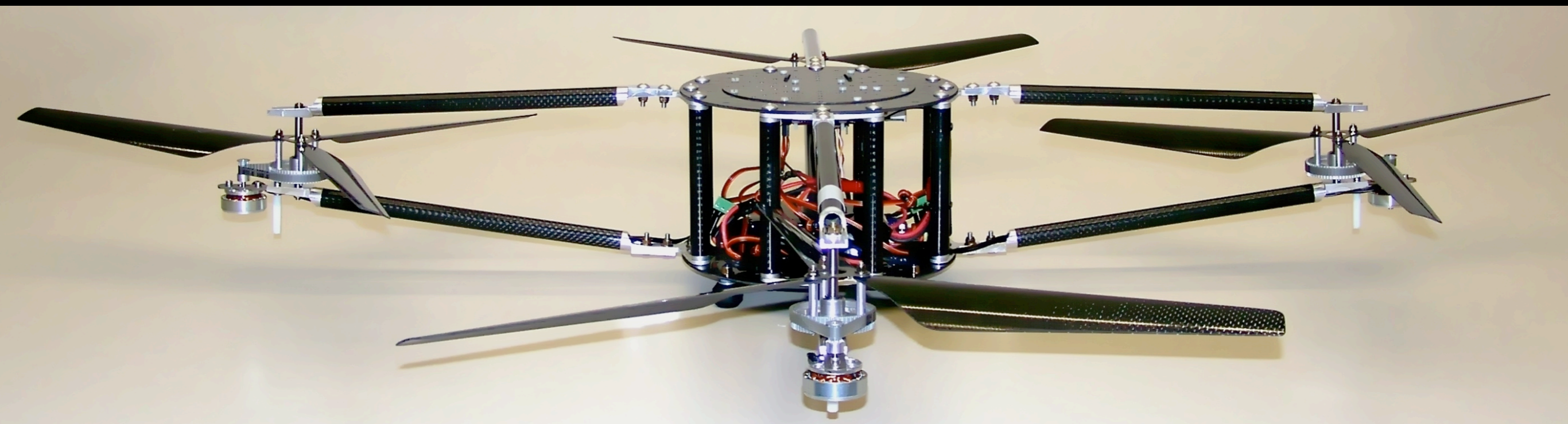
Christoph Kirsch  
Universität Salzburg



EPFL Seminar  
May 2008

# [tiptoe.cs.uni-salzburg.at](http://tiptoe.cs.uni-salzburg.at)

- Silviu Craciunas\* (Programming Model)
- Hannes Payer\* (Memory Management)
- Harald Röck (VM, Scheduling)
- Ana Sokolova\* (Theoretical Foundation)
- Horst Stadler (I/O Subsystem)

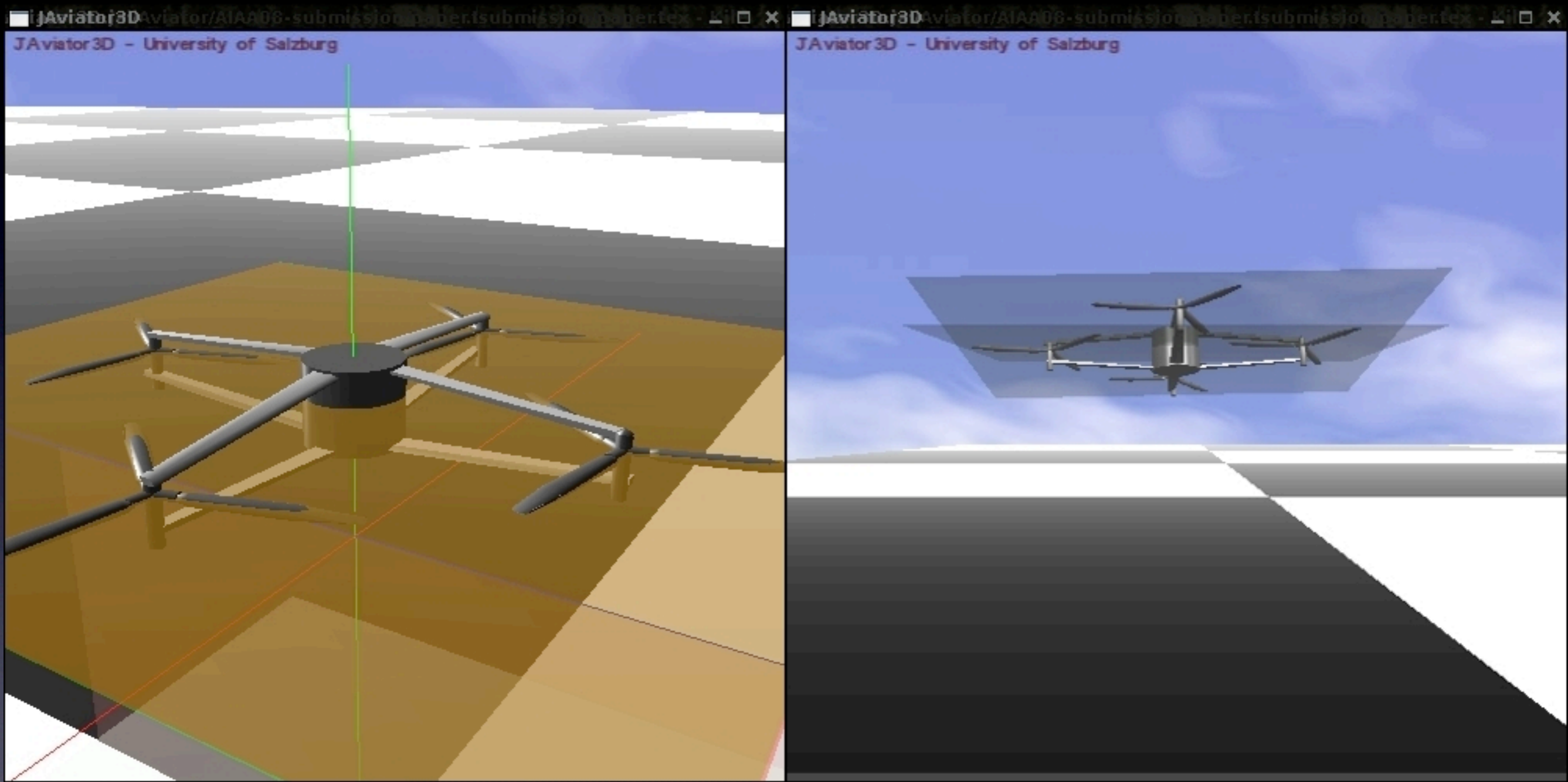


# The JAviator

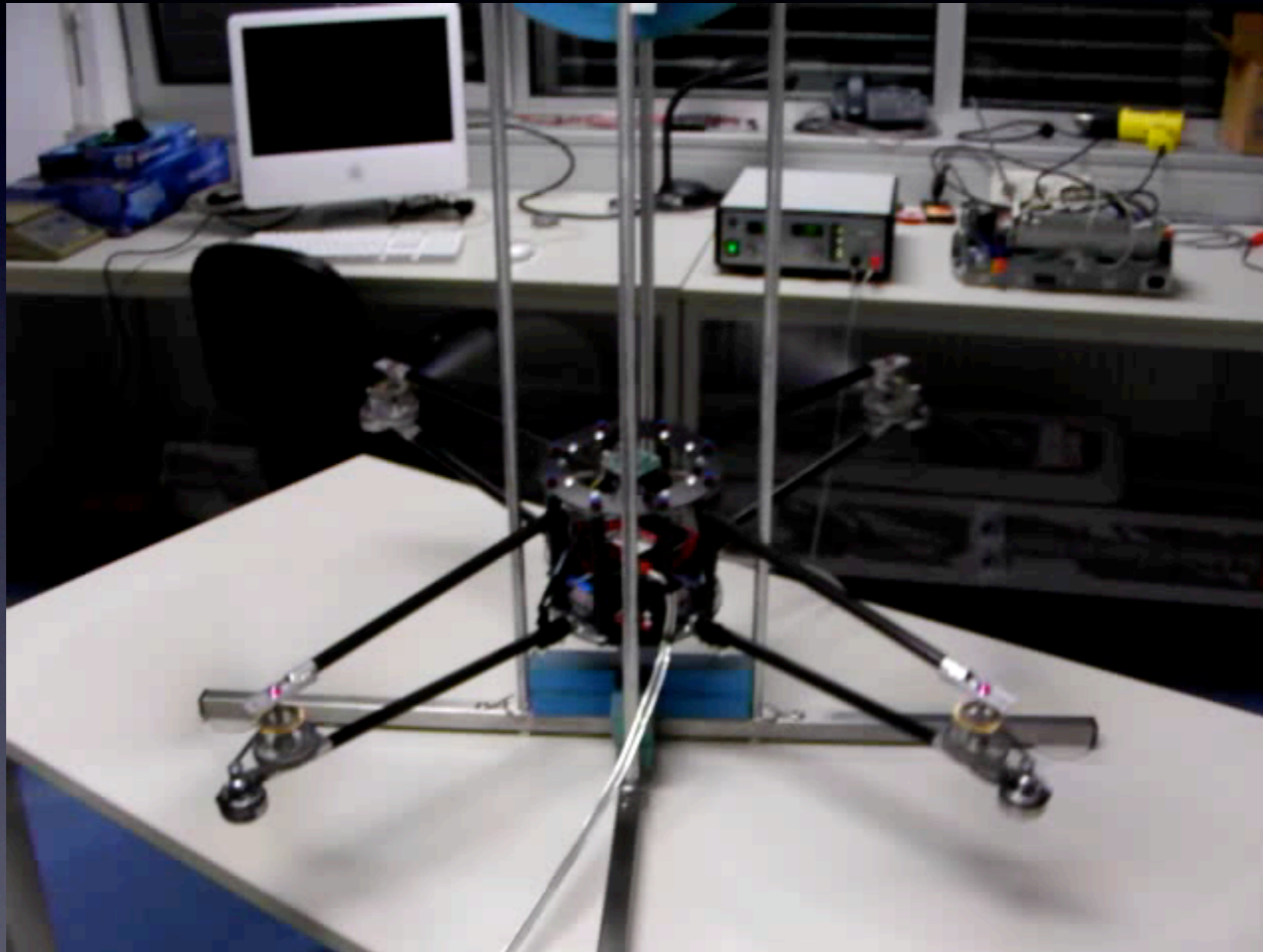
[javiator.cs.uni-salzburg.at](http://javiator.cs.uni-salzburg.at)

# Quad-Rotor Helicopter





# Flight Control



# Free Flight



# “Theorem”

- (Compositionality) The **time** and **space** a software process needs to execute is determined by the **process**, not the system and not other software processes.
- (Predictability) The **system** can tell how much **time** and **space** is available without looking at any existing software processes.



# “Corollary”

- **(Memory)** The time a software process takes to **allocate** and **free** a memory object is determined by the size of the **object**.
- **(I/O)** The time a software process takes to **read** input data and **write** output data is determined by the size of the **data**.

# Outline

1. Programming Model
2. Concurrency Management
3. Memory Management
4. I/O Management

# State of the Art

- Traditional real-time process model:
  - A set of periodic tasks with **deadlines**
- Synchronous reactive programs
- Logical execution time (LET) model
  - A set of periodic tasks with deterministic **input** and **output times**

# Compositionality

- System of tasks with deadlines:
  - Existing tasks still meet **deadlines** even when adding/removing tasks
- System of LET tasks:
  - Existing tasks maintain **input** and **output times** even when adding/removing tasks

# Tiptoe Process Model

- Tiptoe processes invoke **process actions**
- **Process actions** are system calls and procedure calls but also just code, which may have optional **workload parameters**
- **Workload parameters** determine the amount of work involved in executing process actions

# Example

- Consider a process that **reads** a video stream from a network connection, **compresses** it, and **stores** it on disk, all in real time
- The process periodically **adapts** the frame rate, **allocates** memory, **receives** frames, **compresses** them, **writes** the result to disk, and finally **deallocates** memory to prepare for the next iteration

# Pseudo Code

```
loop {  
    int number_of_frames = determine_rate();  
  
    allocate_memory(number_of_frames);  
    read_from_network(number_of_frames);  
  
    compress_data(number_of_frames);  
  
    write_to_disk(number_of_frames);  
    deallocate_memory(number_of_frames);  
} until (done);
```

# Tiptoe Programming Model

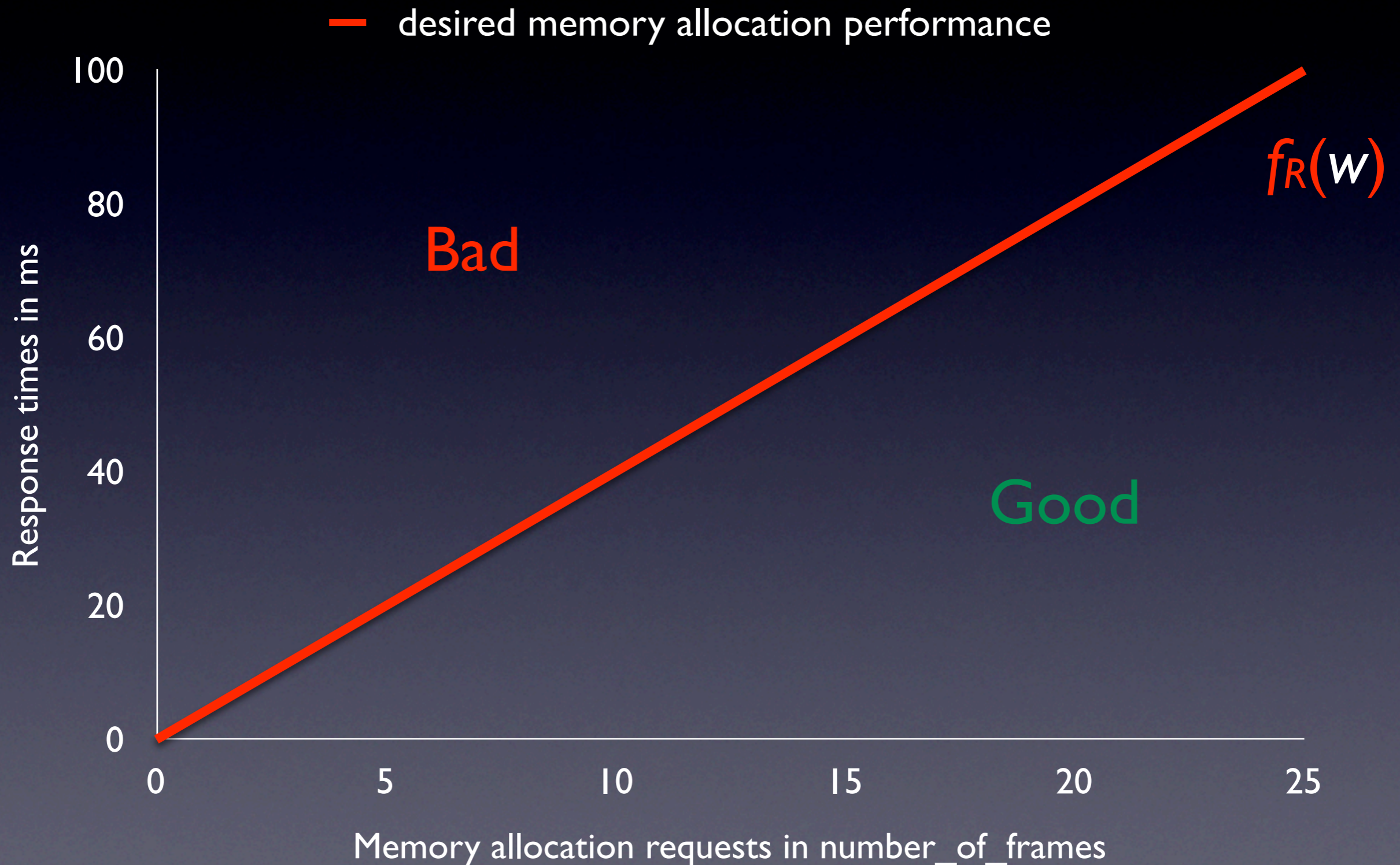
- Process actions are characterized by their **execution time** and **response time** in terms of their workload parameters
- The **execution time** is the time it takes to execute an action in the absence of concurrent activities
- The **response time** is the time it takes to execute an action in the presence of concurrent activities



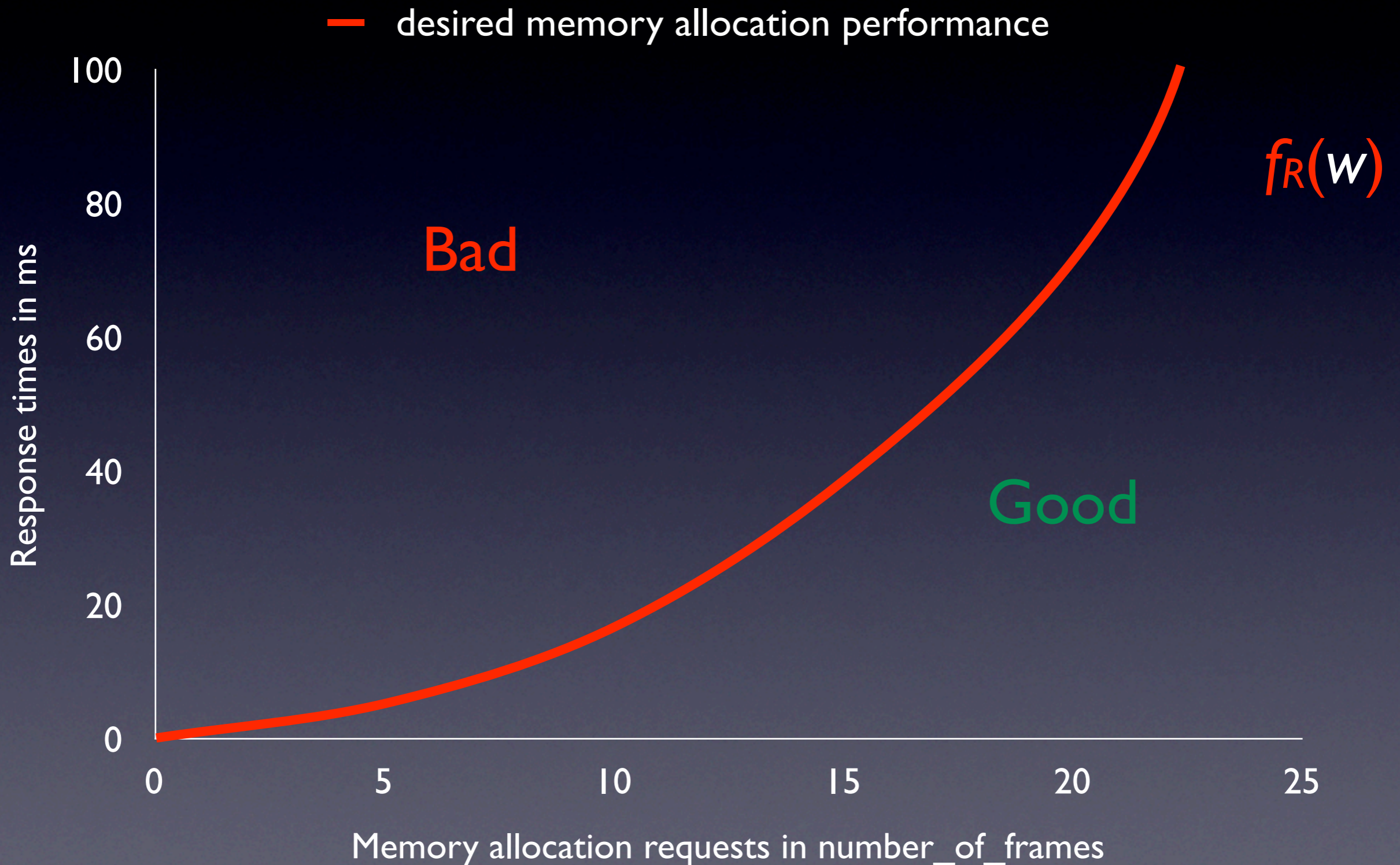
# Compositionality

- System of Tiptoe processes:
  - The individual actions of running Tiptoe processes maintain their **response times** even when adding/removing processes

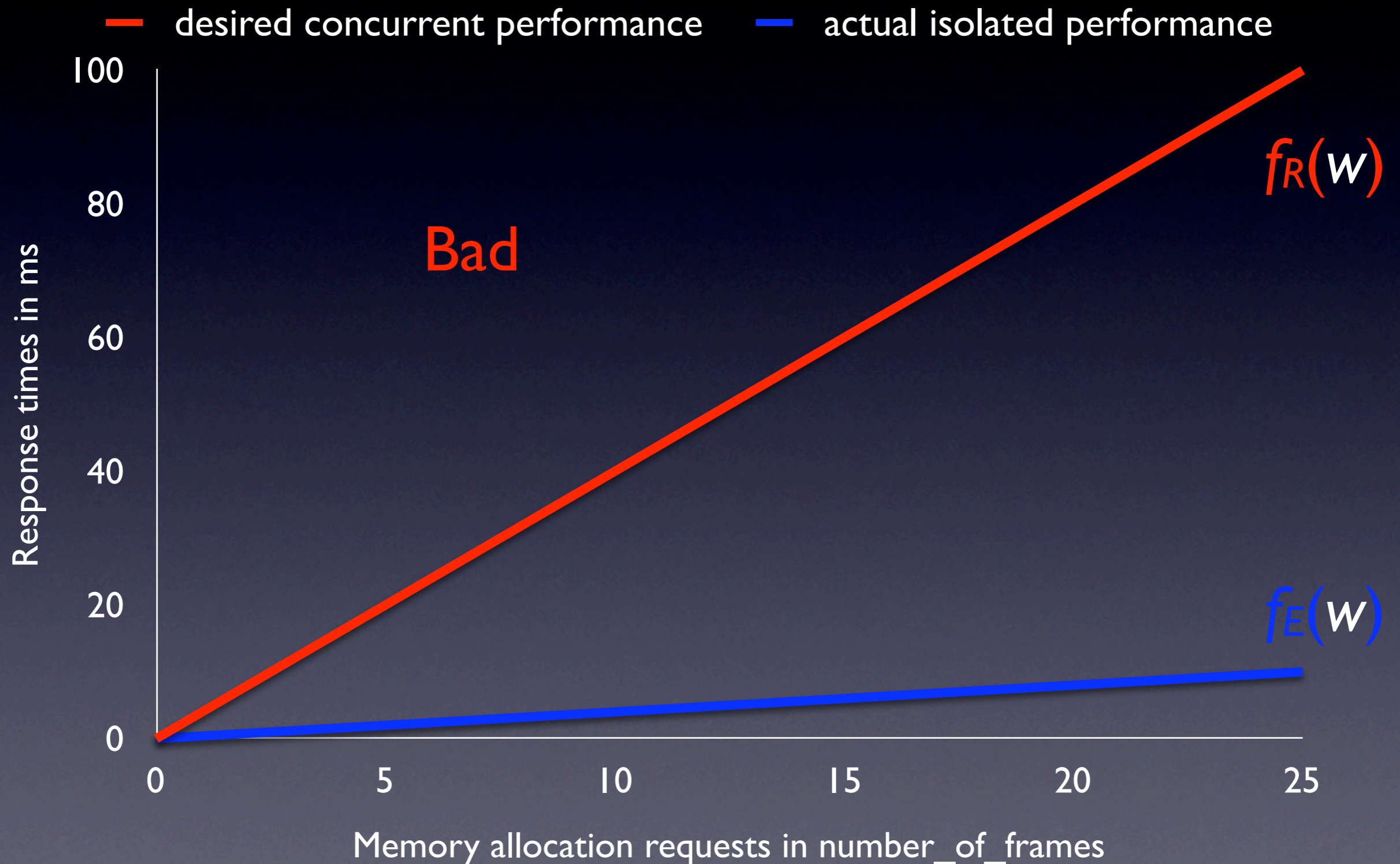
# Response-Time Function



# Compositional Response!



# Execution-Time Function



# Utilization Function:

$$f_U(\mathbf{w}) = \frac{f_E(\mathbf{w})}{f_R(\mathbf{w})}$$

With

$$f_R(w) = 4 * w \text{ (in ms)}$$

$$f_E(w) = 0.4 * w \text{ (in ms)}$$

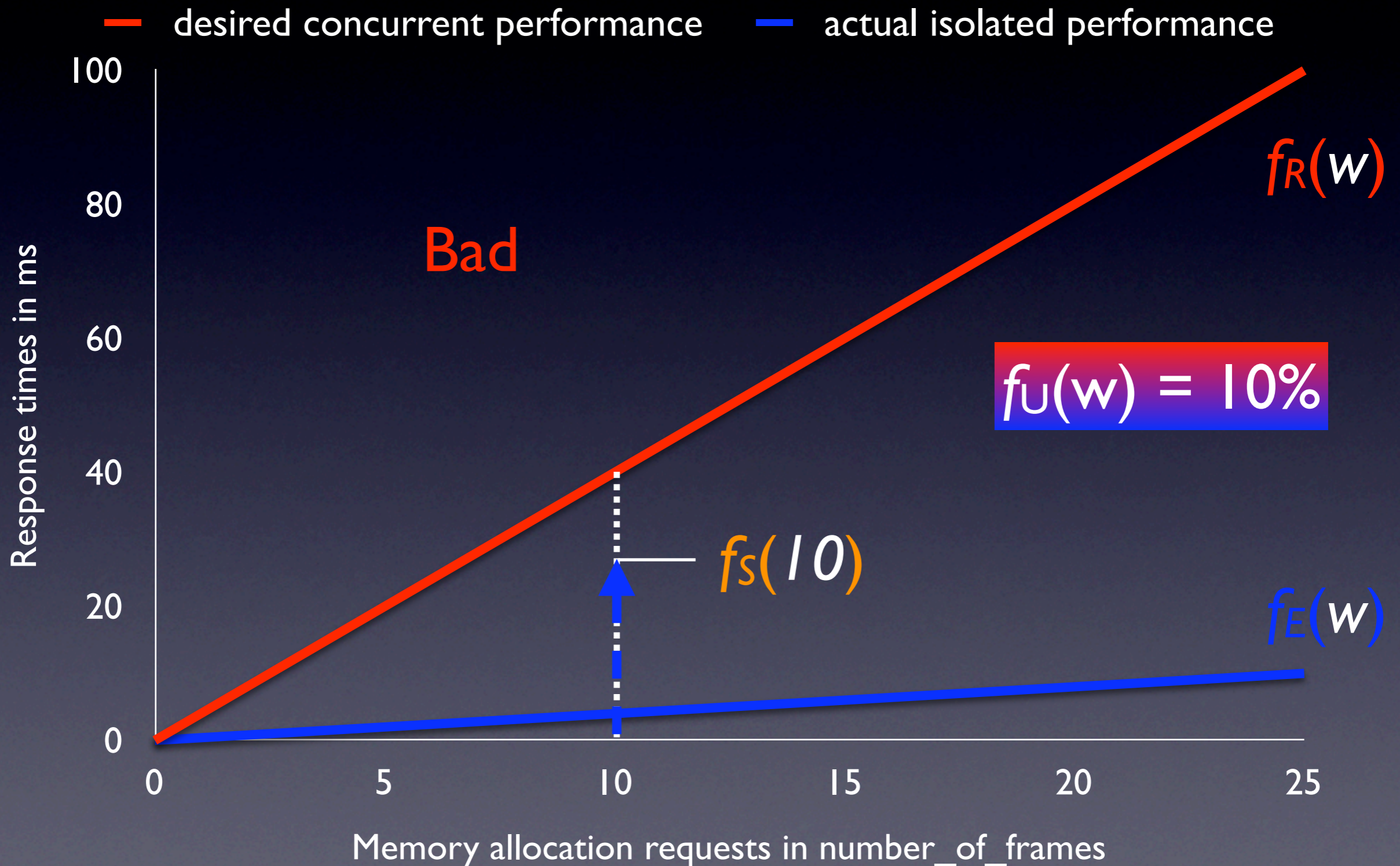
we have that

$$f_U(w) = 10\% \text{ (for } w > 0)$$

# Outline

1. Programming Model
2. **Concurrency Management**
3. Memory Management
4. I/O Management

# Scheduled Response Time





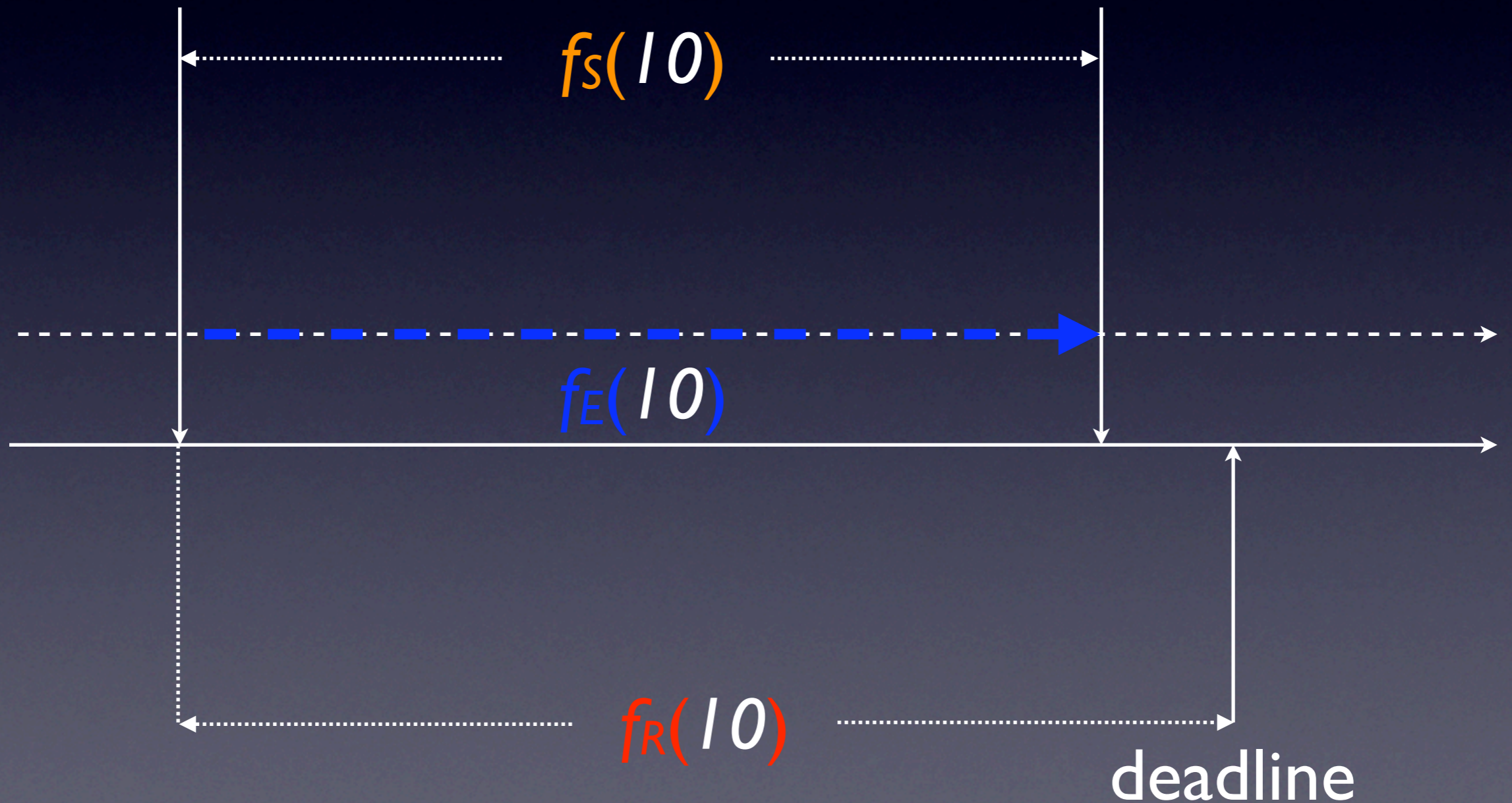
# Scheduling and Admission

- Process scheduling:
  - How do we efficiently **schedule** processes on the level of individual process actions?
- Process admission:
  - How do we efficiently **test** schedulability of newly arriving processes

# Just use EDF, or not?

action arrives

action completes

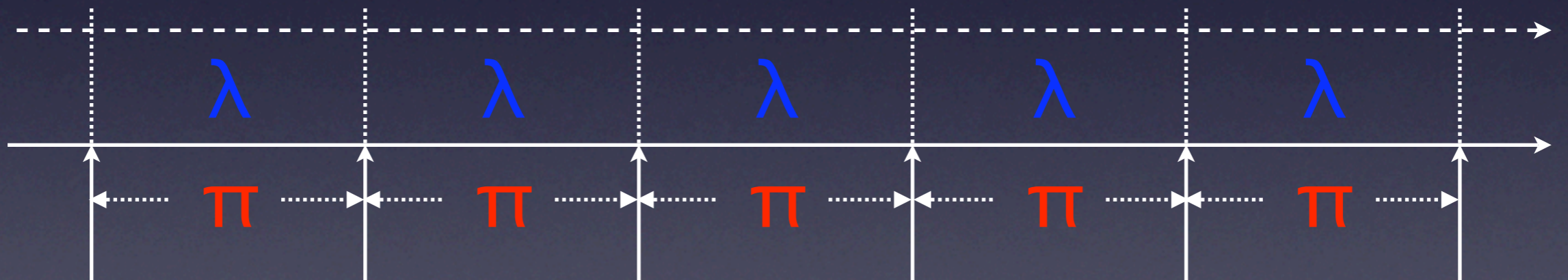


# Virtual Periodic Resource

limit:  $\lambda$

period:  $\pi$

utilization:  $\lambda / \pi$



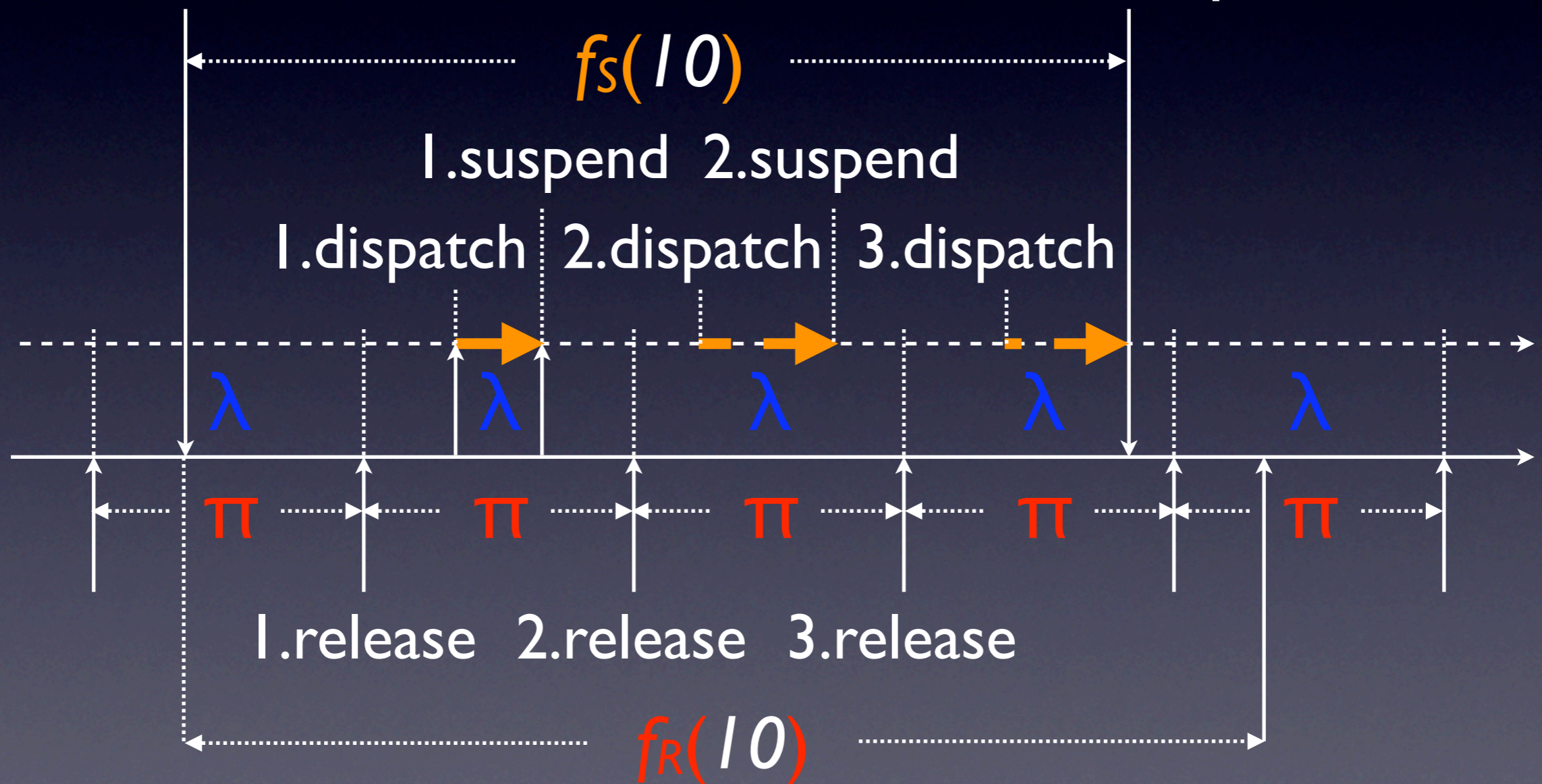
# Tiptoe Process Model

- Each Tiptoe process declares a finite set of **virtual periodic resources**
- Each process action of a Tiptoe process uses exactly one **virtual periodic resource** declared by the process

# Release, Dispatch, Suspend

action arrives

action completes



# Scheduling Strategies

- release action upon arrival at the beginning of next period (**release strategy**)
- dispatch released actions in EDF order using periods as deadlines (**dispatch strategy**)
- suspend running actions until beginning of next period when limit is exhausted (**limit strategy**)

$$\forall w \in E_D. f_S(w) \leq f_R(w) ?$$

$$\forall w \in E_D.$$

$$f_E(w) + (\pi - \lambda) * (\lceil f_E(w) / \lambda \rceil - 1)$$

$$\leq f_S(w) \leq$$

$$(\pi - 1) + \pi * (\lceil f_E(w) / \lambda \rceil - 1) + \pi$$

if

$$\sum_P \max_R (\lambda_{PR} / \pi_{PR}) \leq 1$$



# Tiptoe Compositionality

$$\forall f_s, f_{s'}. \forall w \in E_D.$$

$$0 \leq | f_s(w) - f_{s'}(w) | \leq 2\pi - 2$$

if

$$\sum_P \max_R (\lambda_{PR} / \pi_{PR}) \leq 1$$

# Logical Response Time

worst case (any):  $2\pi - 2$

best case (LRT):  $\pi - 1$

With  $\lambda / \pi = c_U$ , we know that

$$\forall w \in U_D. f_S(w) \leq f_R(w) + \pi$$

if

$\pi$  divides  $f_R(w)$  evenly

and

$$\sum_P \max_R (\lambda_{PR} / \pi_{PR}) \leq 1$$

$$\forall w \in \mathcal{U}_D. f_S(w) \leq f_R(w) + \pi$$

# Utilization Function:

$$f_U(\mathbf{w}) = \frac{f_E(\mathbf{w}) - d_E}{f_R(\mathbf{w}) - d_R}$$

(if  $f_R(\mathbf{w}) > d_R$ )

For example, with

$$f_R(w) = 4 * w + 4 \text{ (in ms)}$$

$$f_E(w) = 0.4 * w + 0.2 \text{ (in ms)}$$

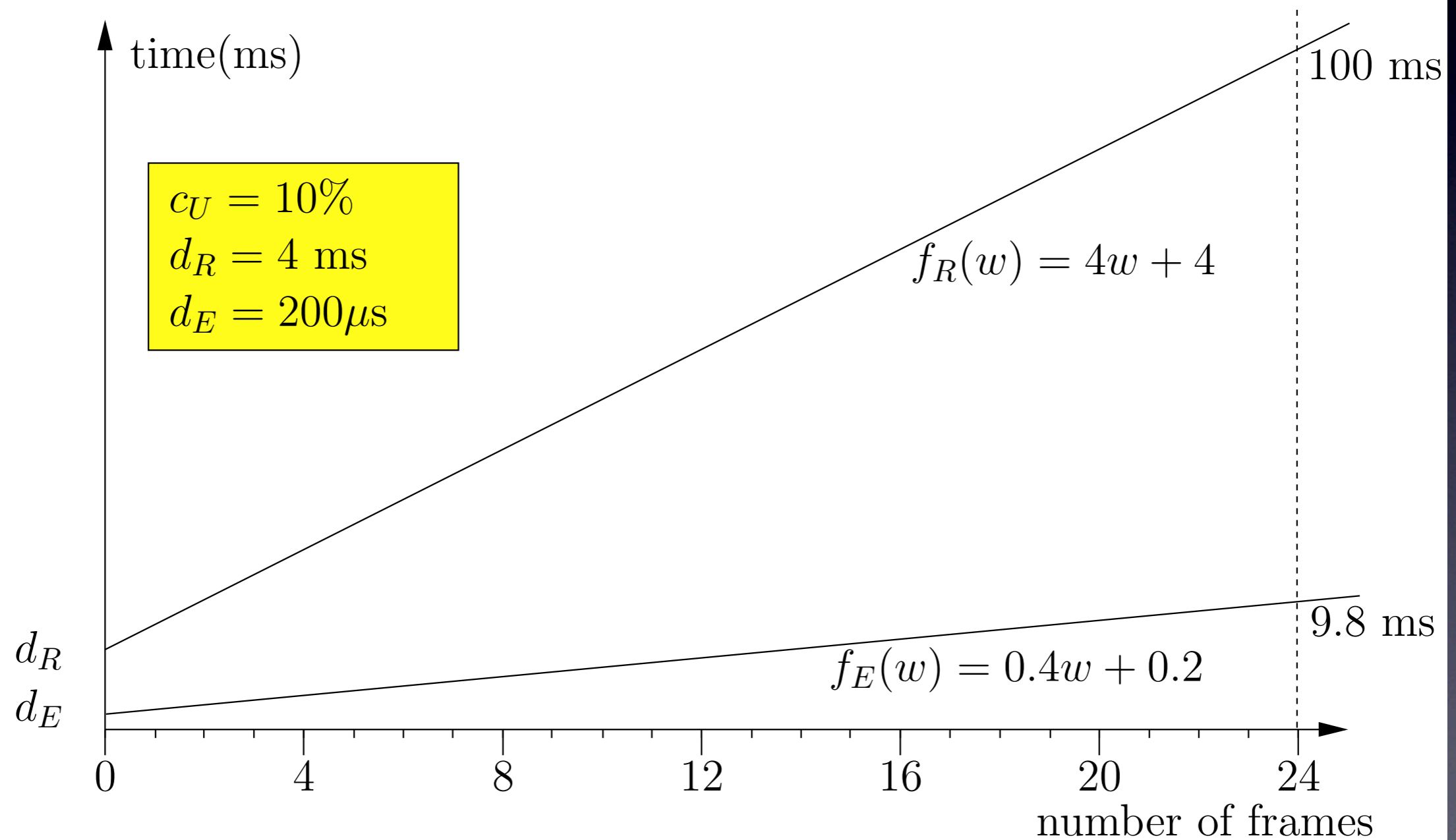
we have again

$$f_U(w) = 10\% \text{ (for } w > 0)$$

$$f_R(1) = 8\text{ms but only } 125\text{fps}$$

$$f_R(24) = 100\text{ms yet } 240\text{fps}$$

# Intrinsic Delay



With  $\lambda / \pi = c_U$ , we know that

$$\forall w \in U_D. f_S(w) \leq f_R(w)$$

if

$$0 < \pi \leq d_R - d_E / c_U, \text{ and}$$

$\pi$  divides  $d_R$  and  $f_R(w) - d_R$  evenly,

$$\text{and } \sum_P \max_R(\lambda_{PR} / \pi_{PR}) \leq 1$$



# Scheduling Algorithm

- maintains a queue of **ready processes** ordered by deadline and a queue of **blocked processes** ordered by release times
- **ordered-insert** processes into queues
- **select-first** processes in queues
- **release** processes by moving and sorting them from one queue to another queue

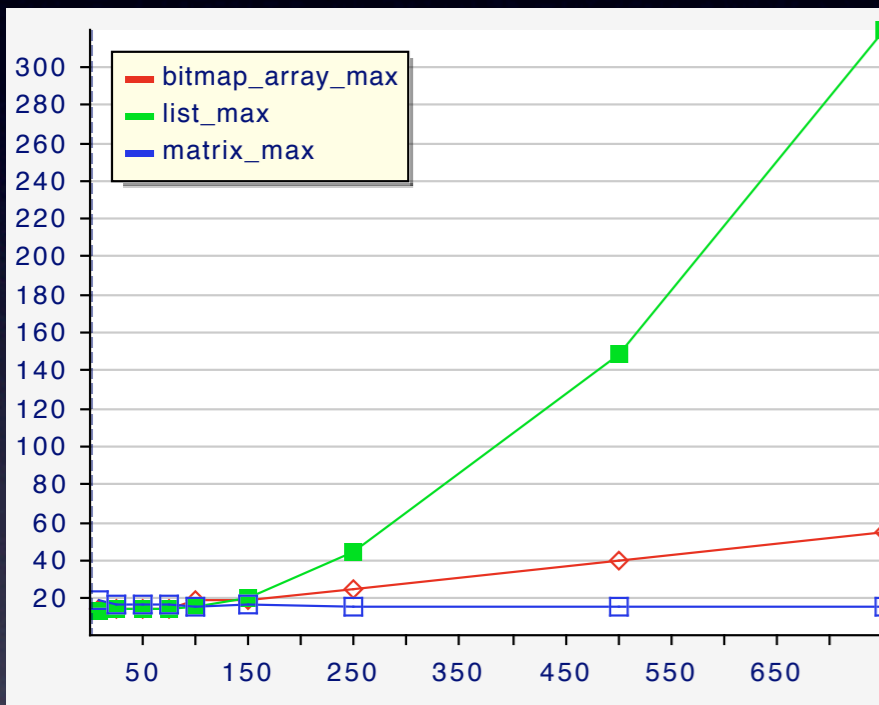
# Time and Space

	list	array	matrix
ordered-insert	$O(n)$	$\Theta(\log(t))$	$\Theta(\log(t))$
select-first	$\Theta(1)$	$O(\log(t))$	$O(\log(t))$
release	$O(n^2)$	$O(\log(t) + n \cdot \log(t))$	$\Theta(t)$

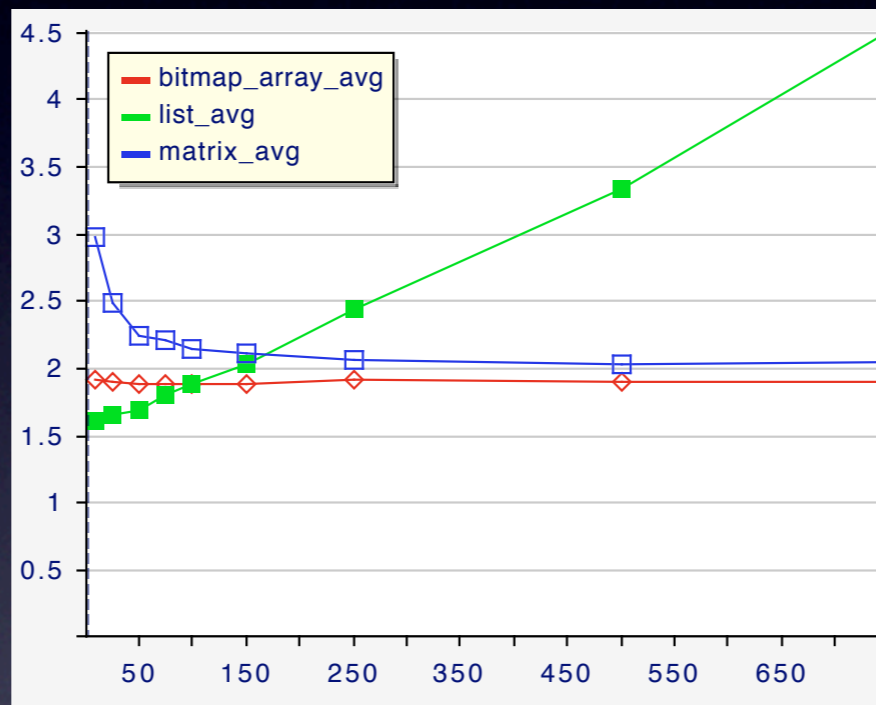
	list	array	matrix
time	$O(n^2)$	$O(\log(t) + n \cdot \log(t))$	$\Theta(t)$
space	$\Theta(n)$	$\Theta(t + n)$	$\Theta(t^2 + n)$

**n: number of processes    t: number of time instants**

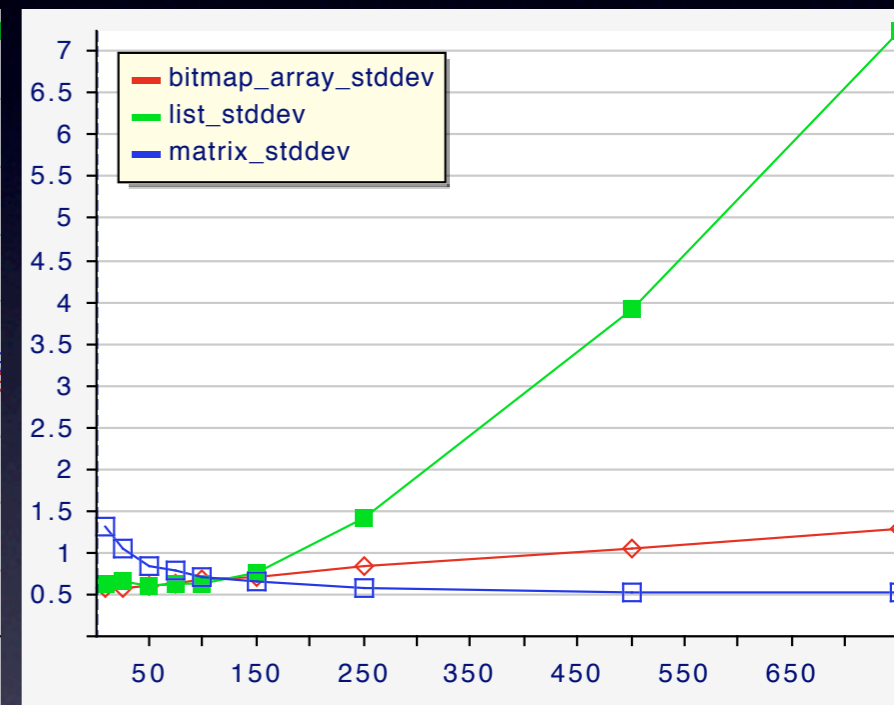
# Scheduler Overhead



Max

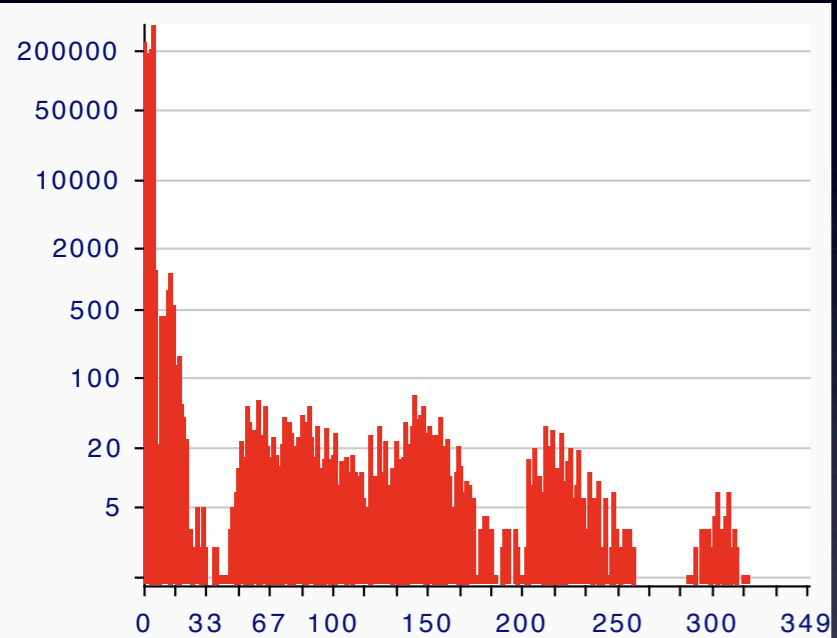


Average

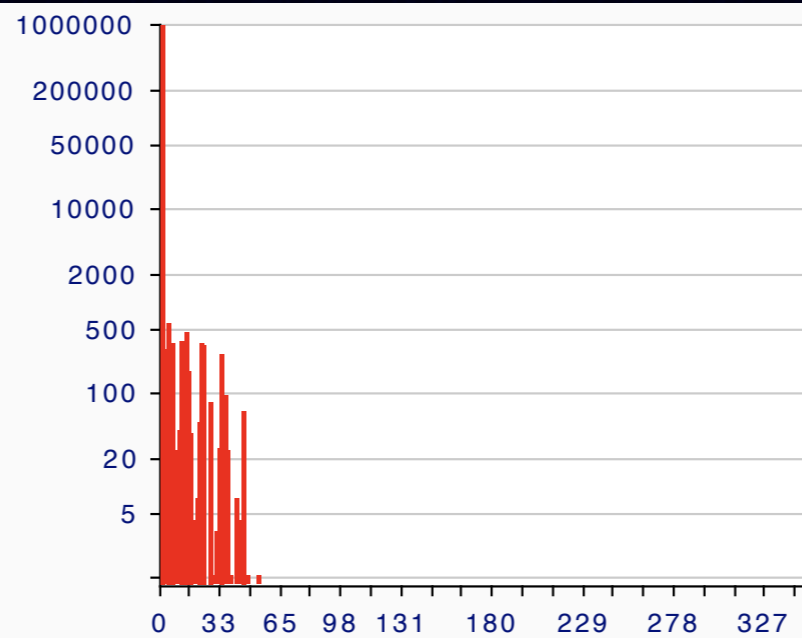


Jitter

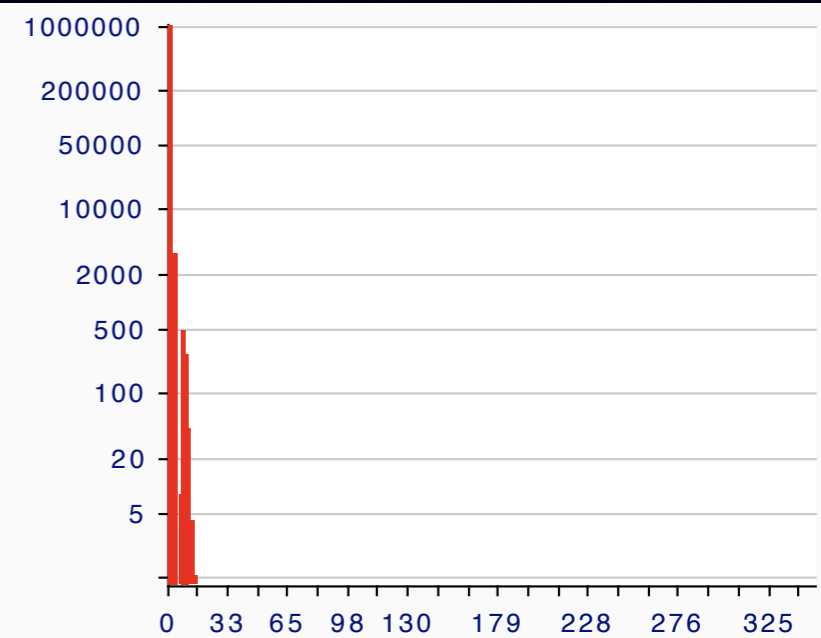
# Execution Time Histograms



List

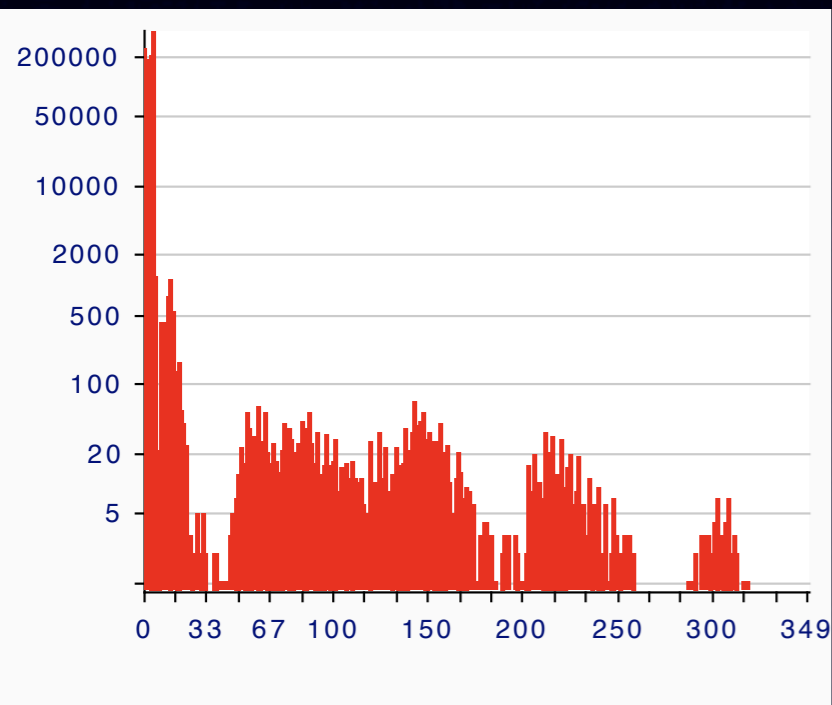


Array

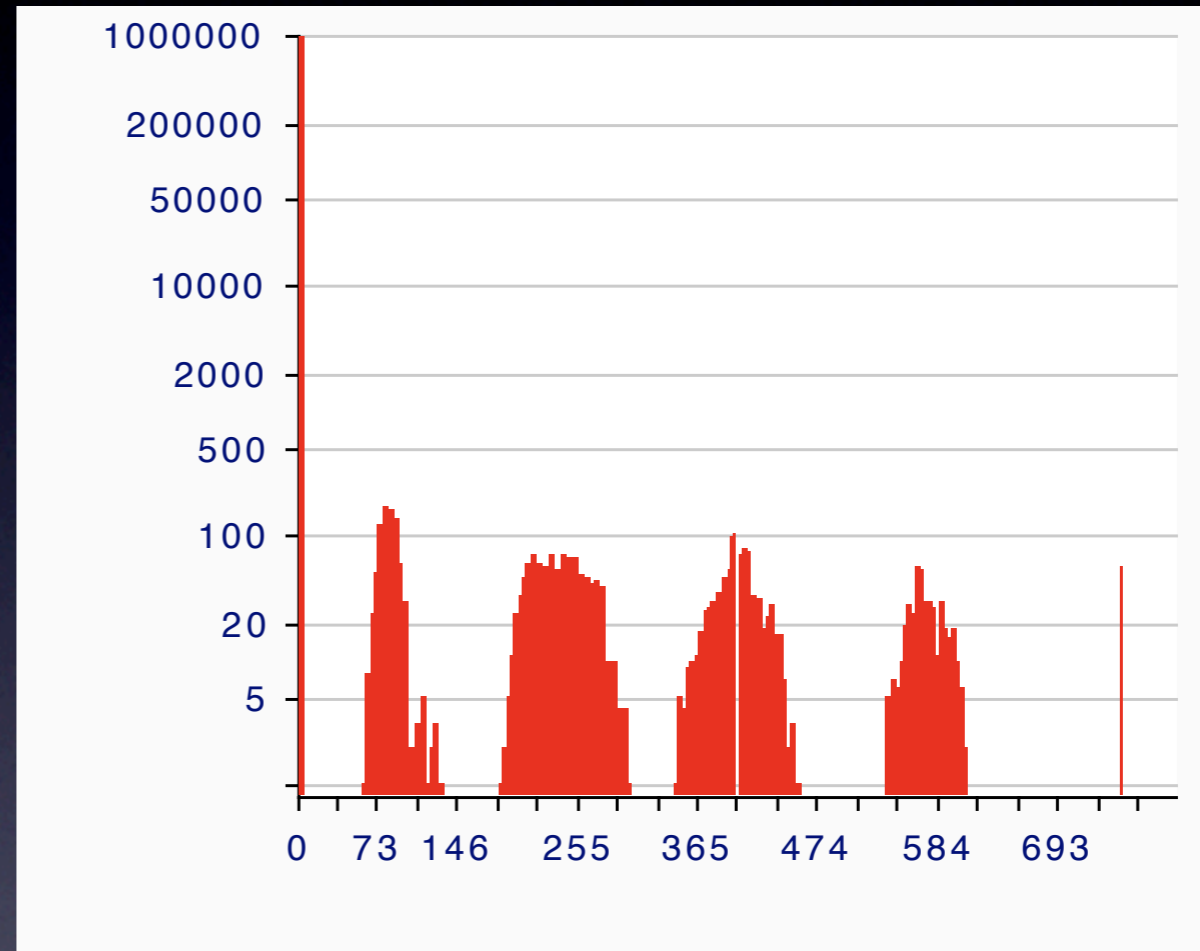


Matrix

# Process Release Dominates

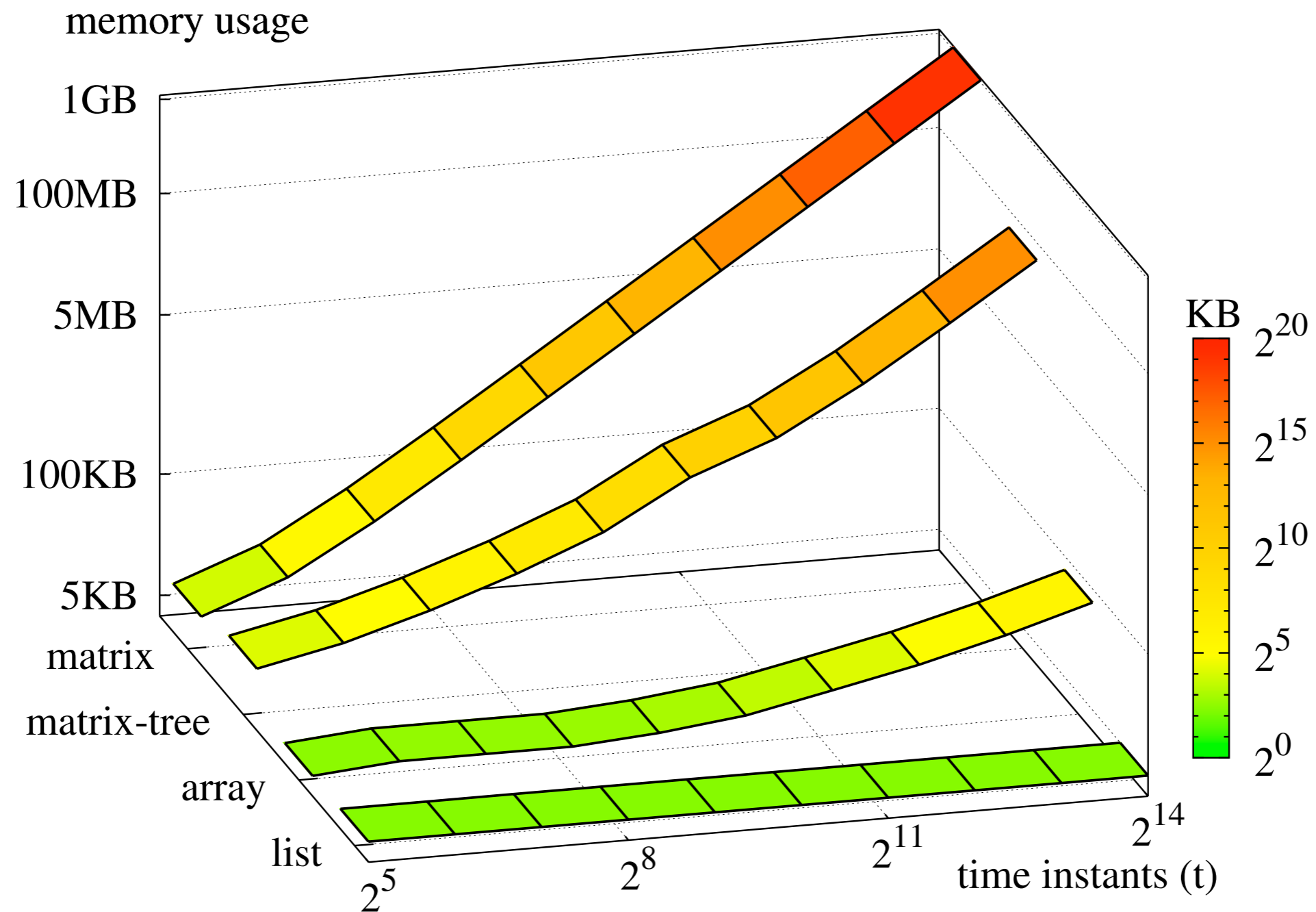


List



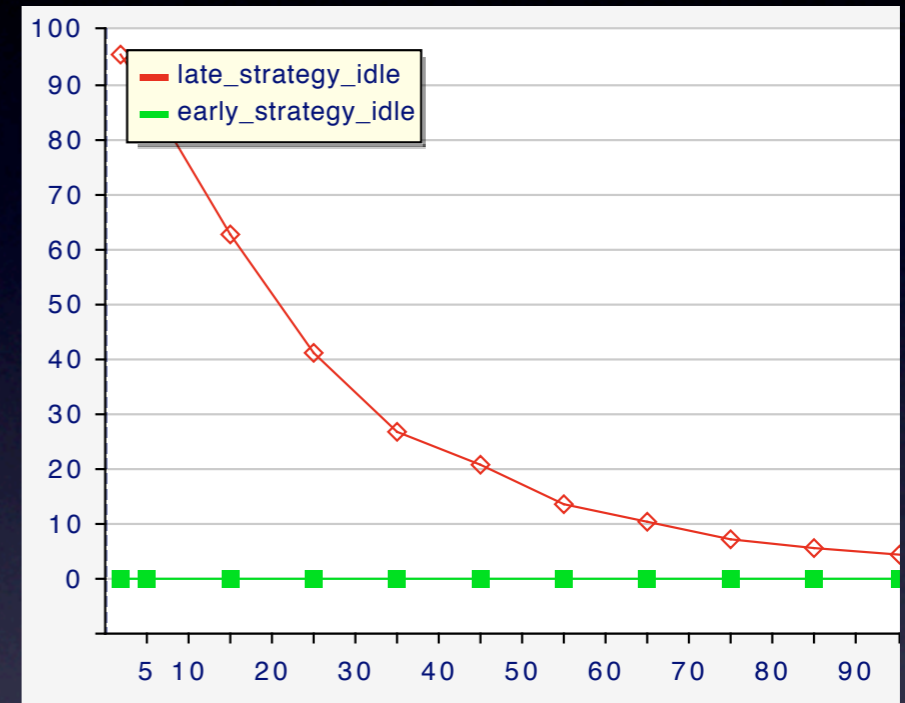
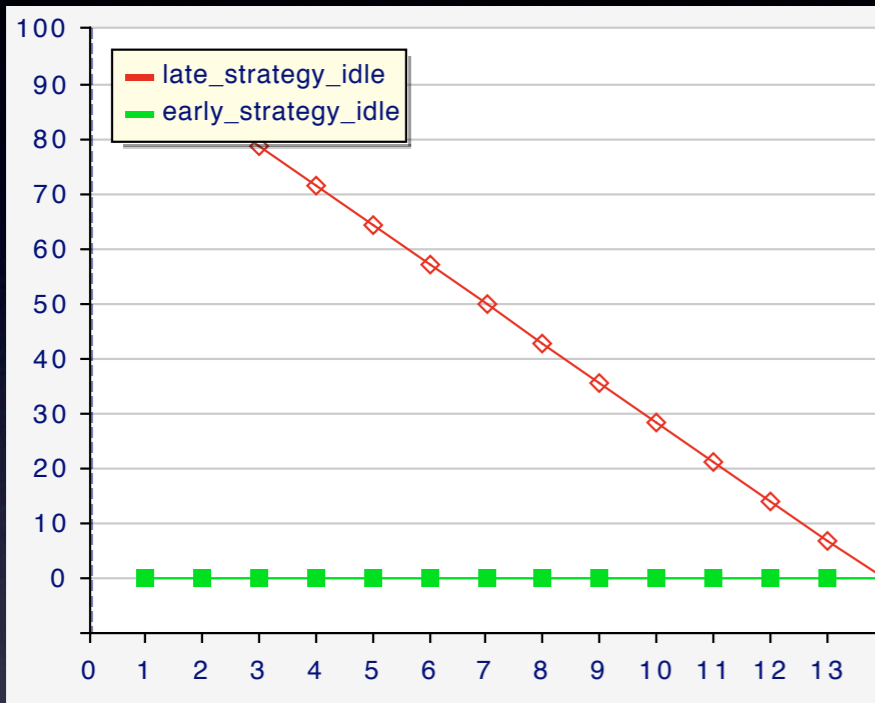
Releases per Instant

# Memory Overhead

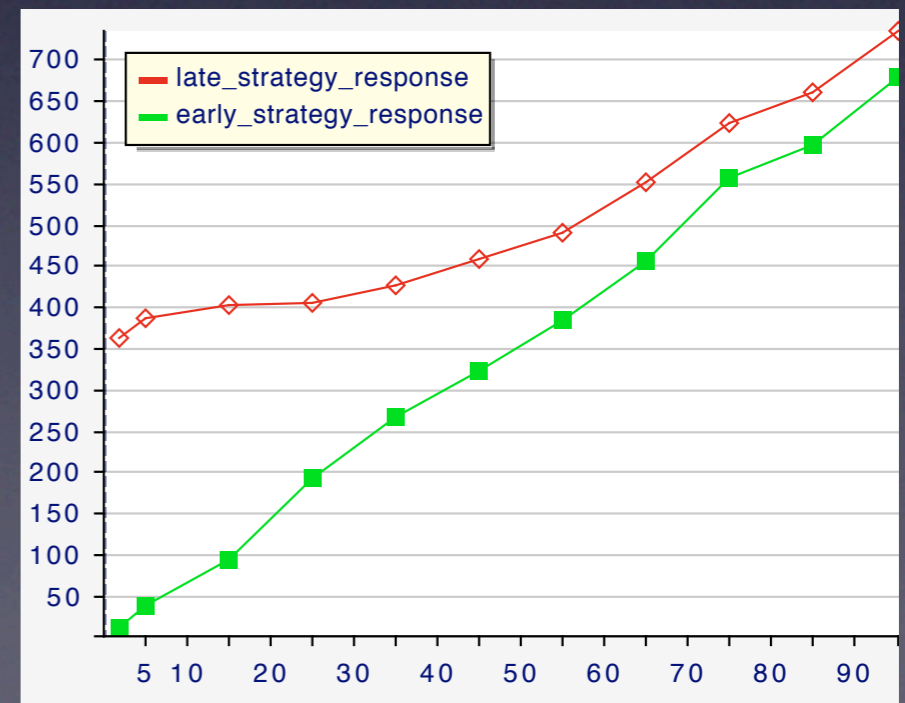
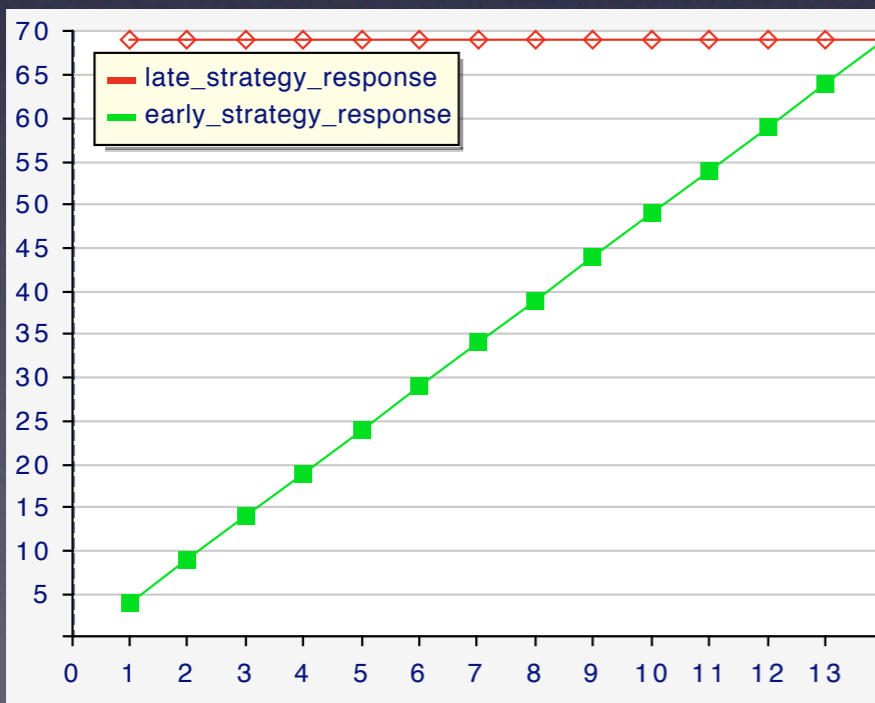


# Release Strategies

Idle Time



Response Time



# Outline

1. Programming Model
2. Concurrency Management
3. **Memory Management**
4. I/O Management



# Outline

1. Programming Model
2. Concurrency Management
3. Memory Management
4. I/O Management

# Current/Future Work

- Concurrent memory management
- I/O subsystem
- Java bytecode VM

Thank you