The JAviator: Time-Portable Programming in Java and C

Christoph Kirsch
Universität Salzburg

Hitachi GST
September 2008

javiator.cs.uni-salzburg.at[#]

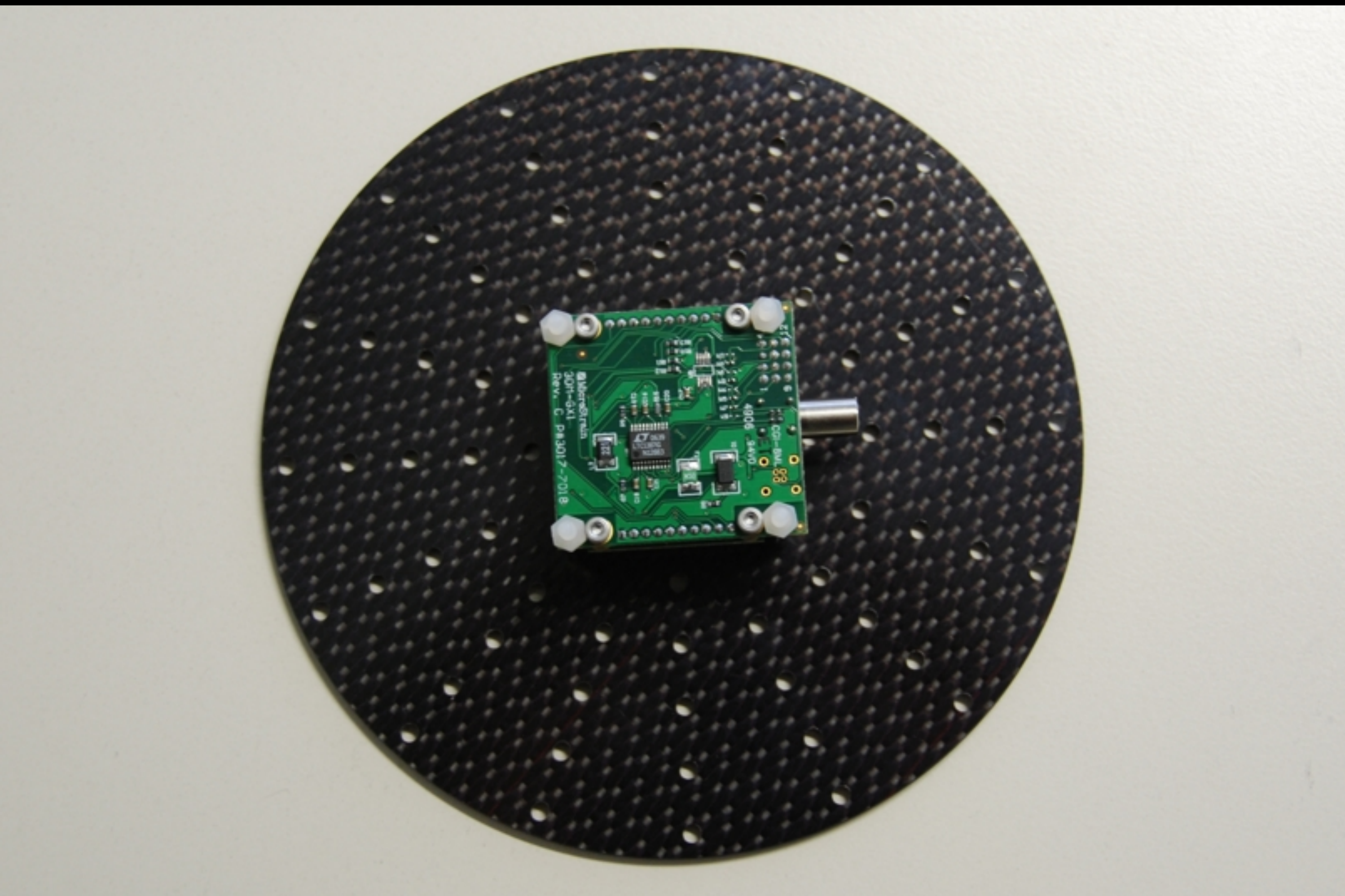
- Silviu Craciunas* (Control Systems)
- Harald Röck (Operating Systems)
- Rainer Trummer (Frame, Electronics)

[#]Supported by a 2007 IBM Faculty Award and the EU ArtistDesign Network of Excellence on Embedded Systems Design
*Supported by Austrian Science Fund Project P18913-N15

The JAviator

javiator.cs.uni-salzburg.at

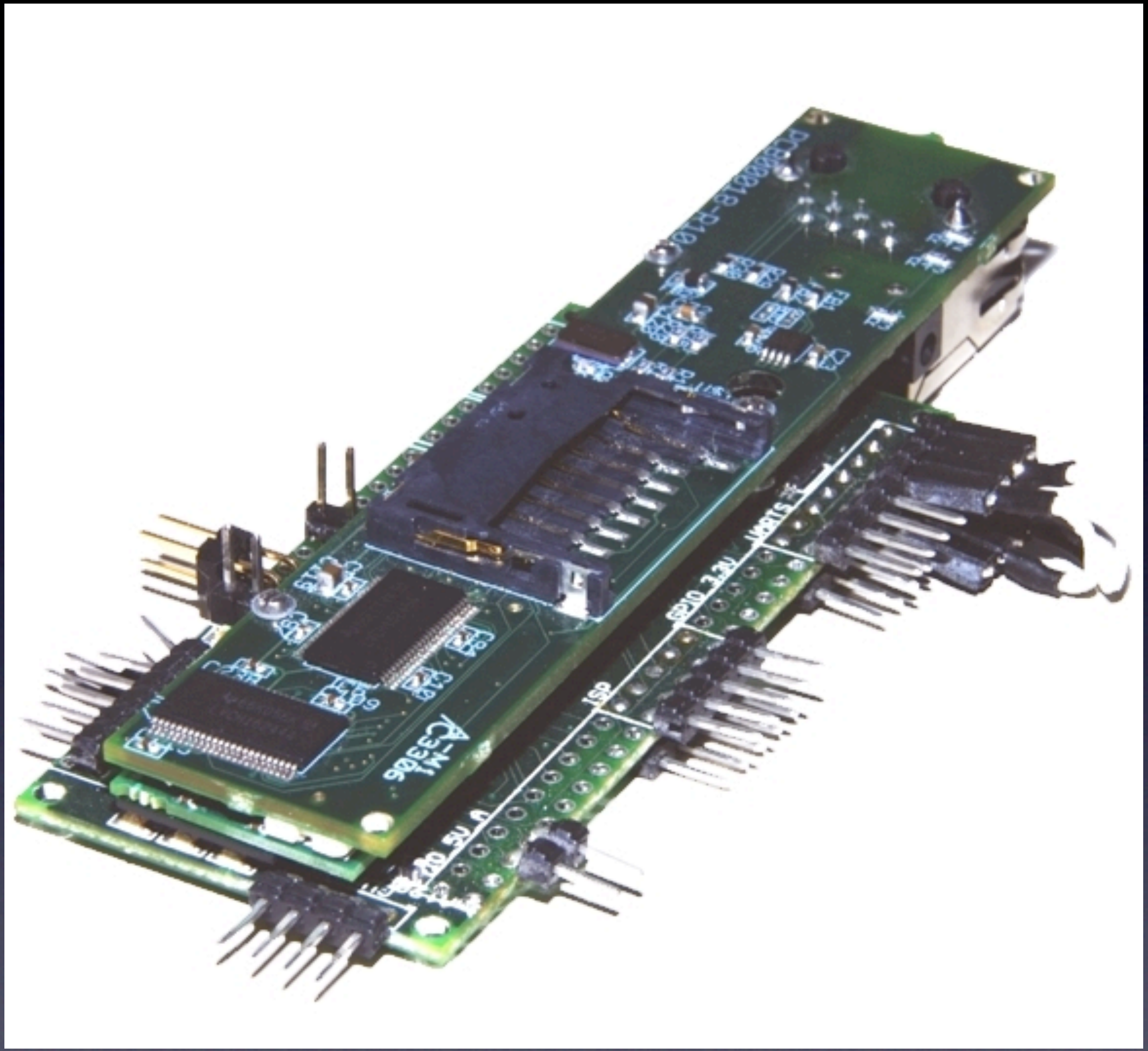
Quad-Rotor Helicopter



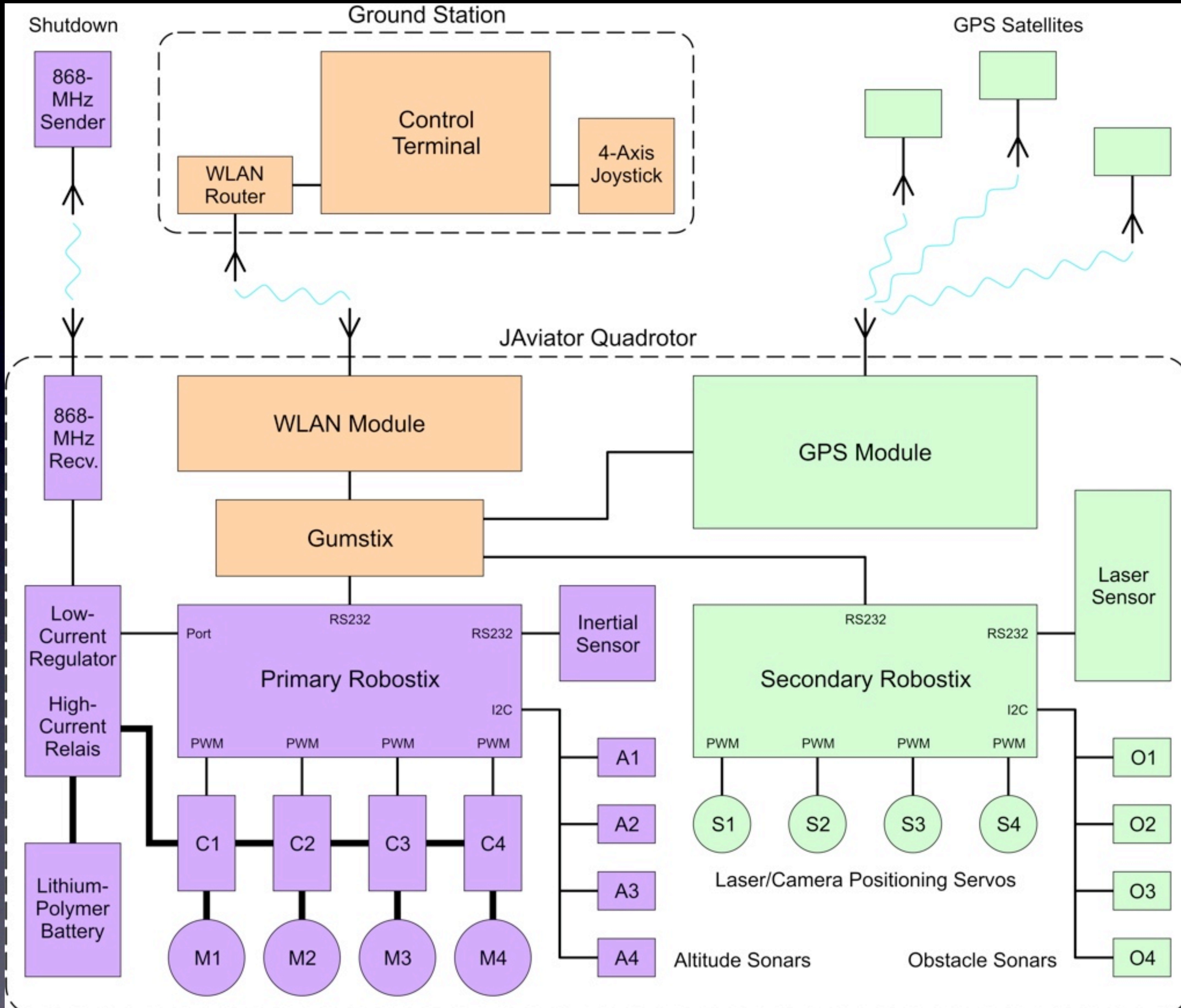
Gyro

Propulsion

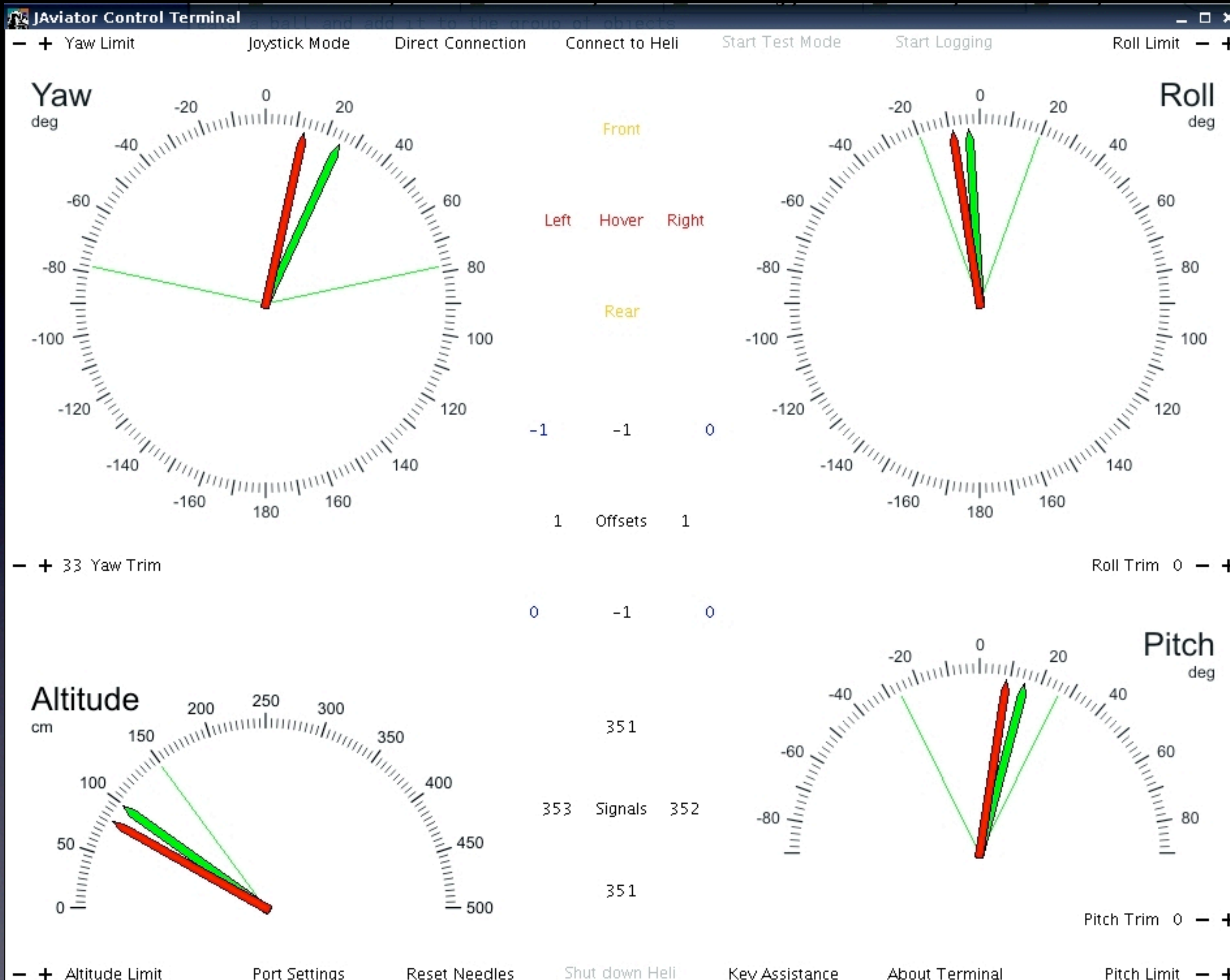
Gumstix

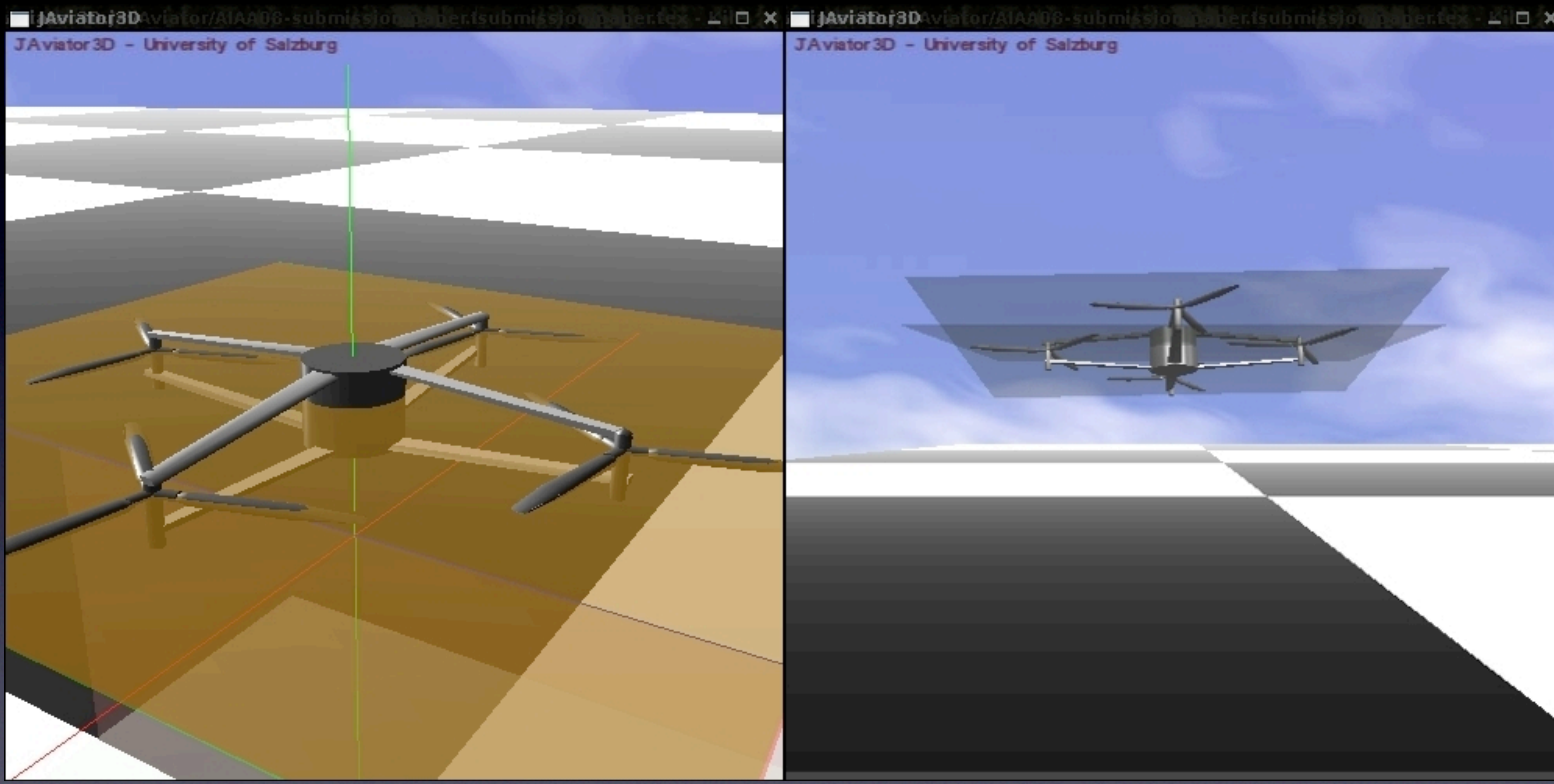


600MHz XScale, 128MB RAM, WLAN, Atmega uController

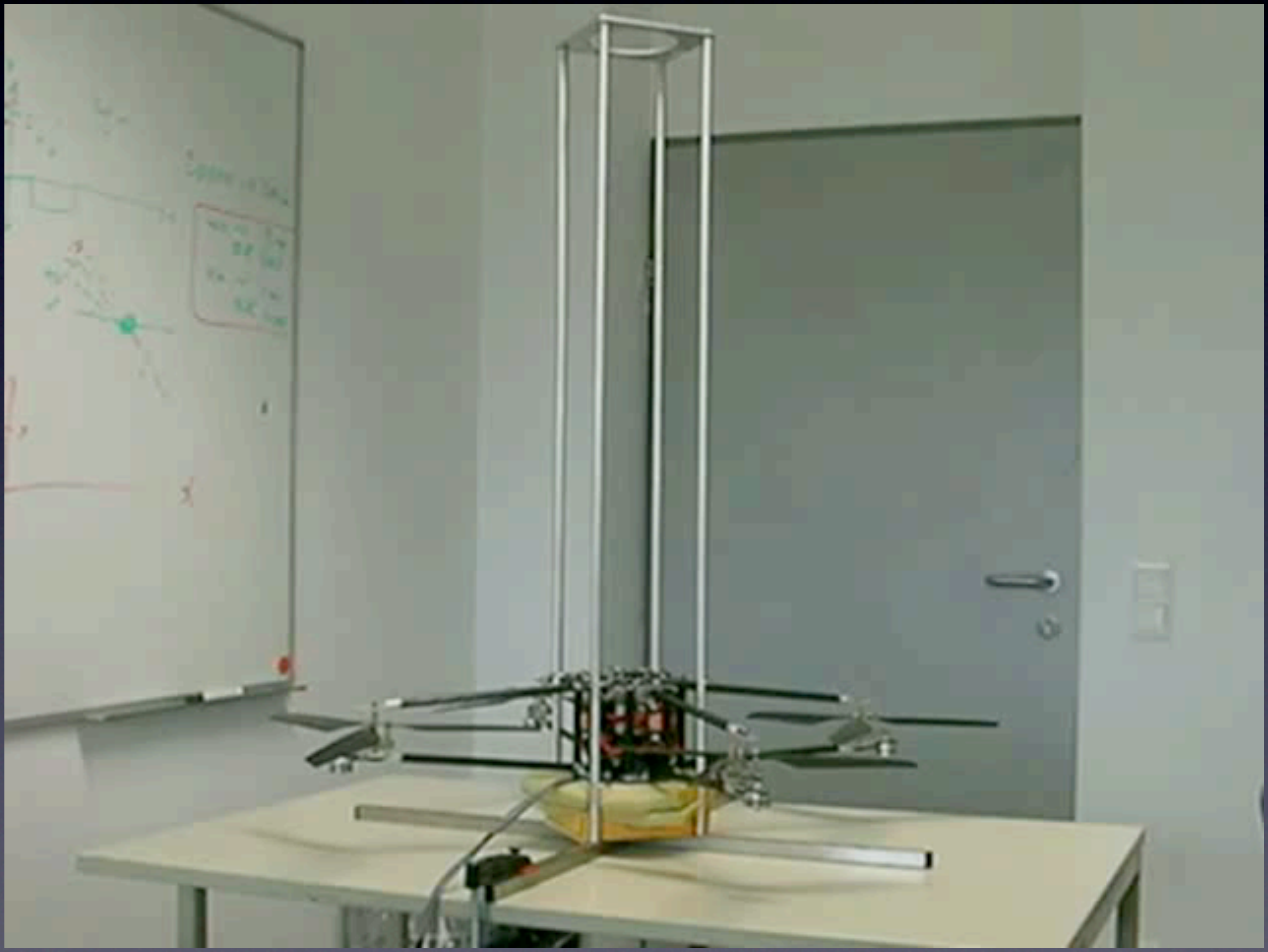


Shutdown
868-MHz Sender
Ground Station
WLAN Router
Control Terminal
4-Axis Joystick
GPS Satellites
JAviator Quadrotor
868-MHz Recv.
WLAN Module
GPS Module
Gumstix
Laser Sensor
Low-Current Regulator
High-Current Relais
Port
RS232
RS232
Primary Robostix
I2C
PWM
PWM
PWM
PWM
Inertial Sensor
RS232
RS232
Secondary Robostix
I2C
PWM
PWM
PWM
PWM
C1
C2
C3
C4
A1
A2
A3
A4
S1
S2
S3
S4
O1
O2
O3
O4
Lithium-Polymer Battery
M1
M2
M3
M4
Laser/Camera Positioning Servos
Altitude Sonars
Obstacle Sonars

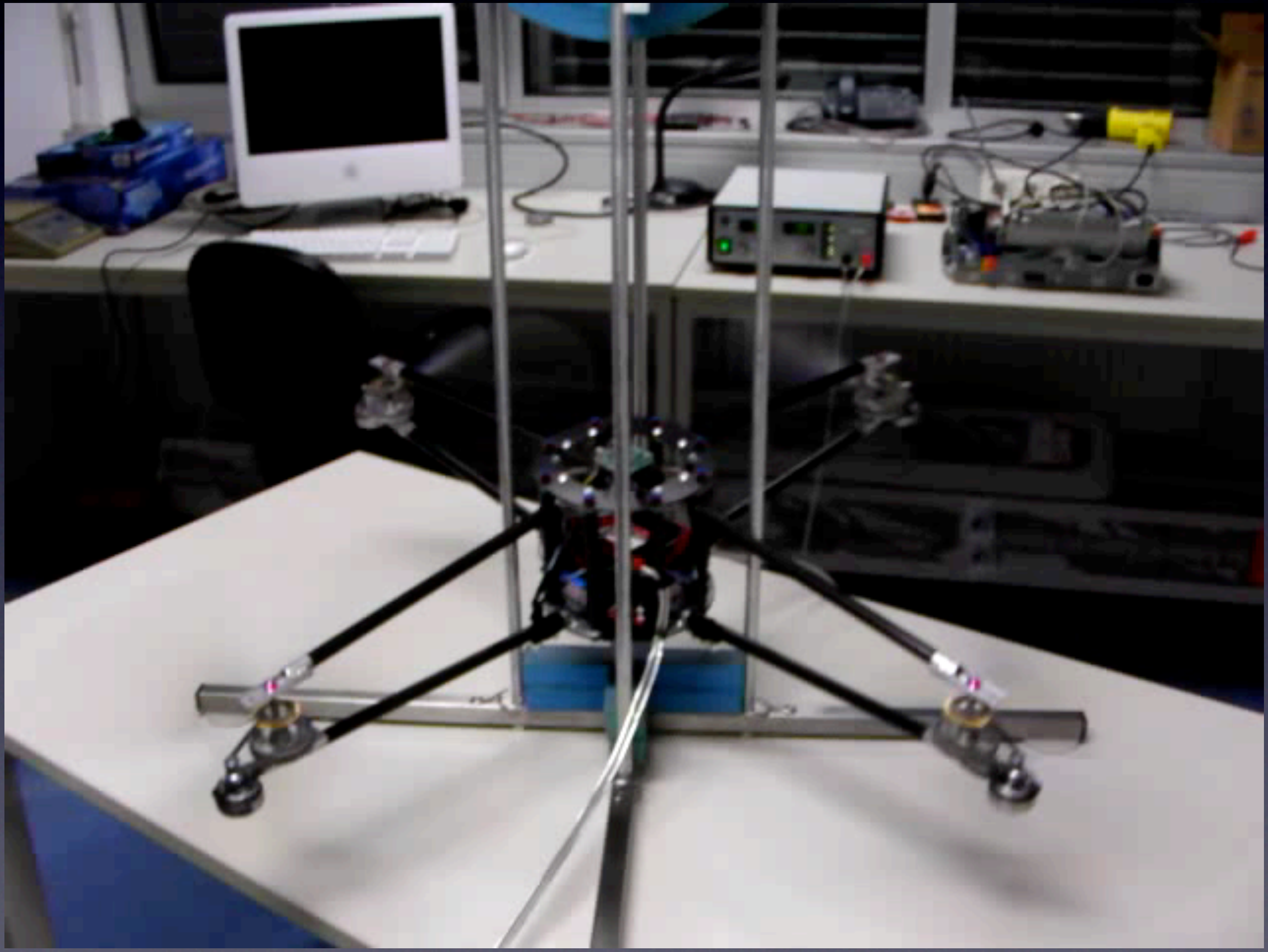

JAviator Control Terminal
Yaw Limit
Joystick Mode
Direct Connection
Connect to Heli
Start Test Mode
Start Logging
Roll Limit
Yaw
deg
Roll
deg
Front
Left
Hover
Right
Rear
-1
-1
0
1
Offsets
1
33 Yaw Trim
Roll Trim 0
0
-1
0
Pitch
deg
Altitude
cm
351
353
Signals
352
351
Pitch Trim 0
Altitude Limit
Port Settings
Reset Needles
Shut down Heli
Key Assistance
About Terminal
Pitch Limit


JAviator3D
JAviator3D - University of Salzburg
JAviator3D
JAviator3D - University of Salzburg

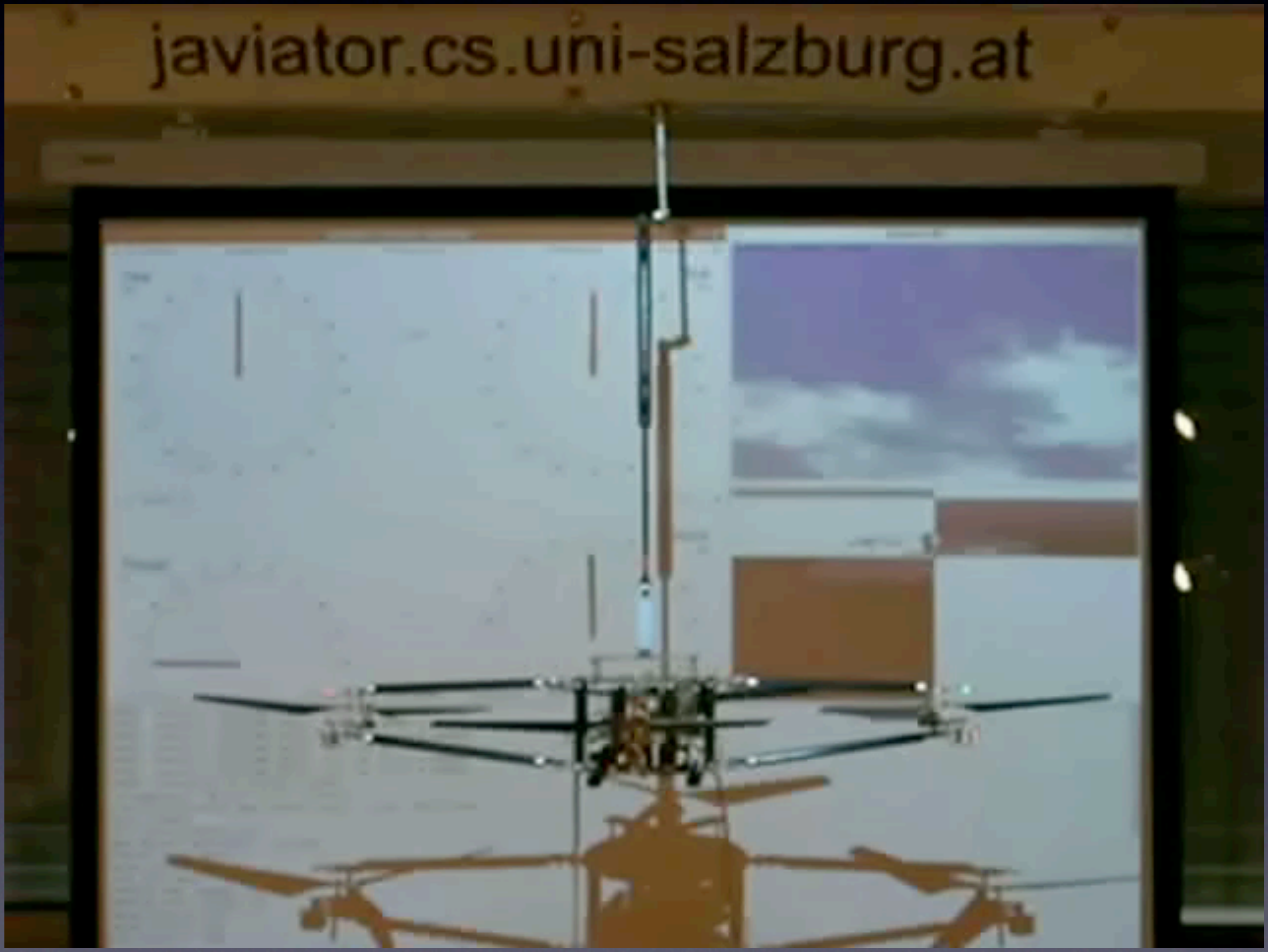
Oops



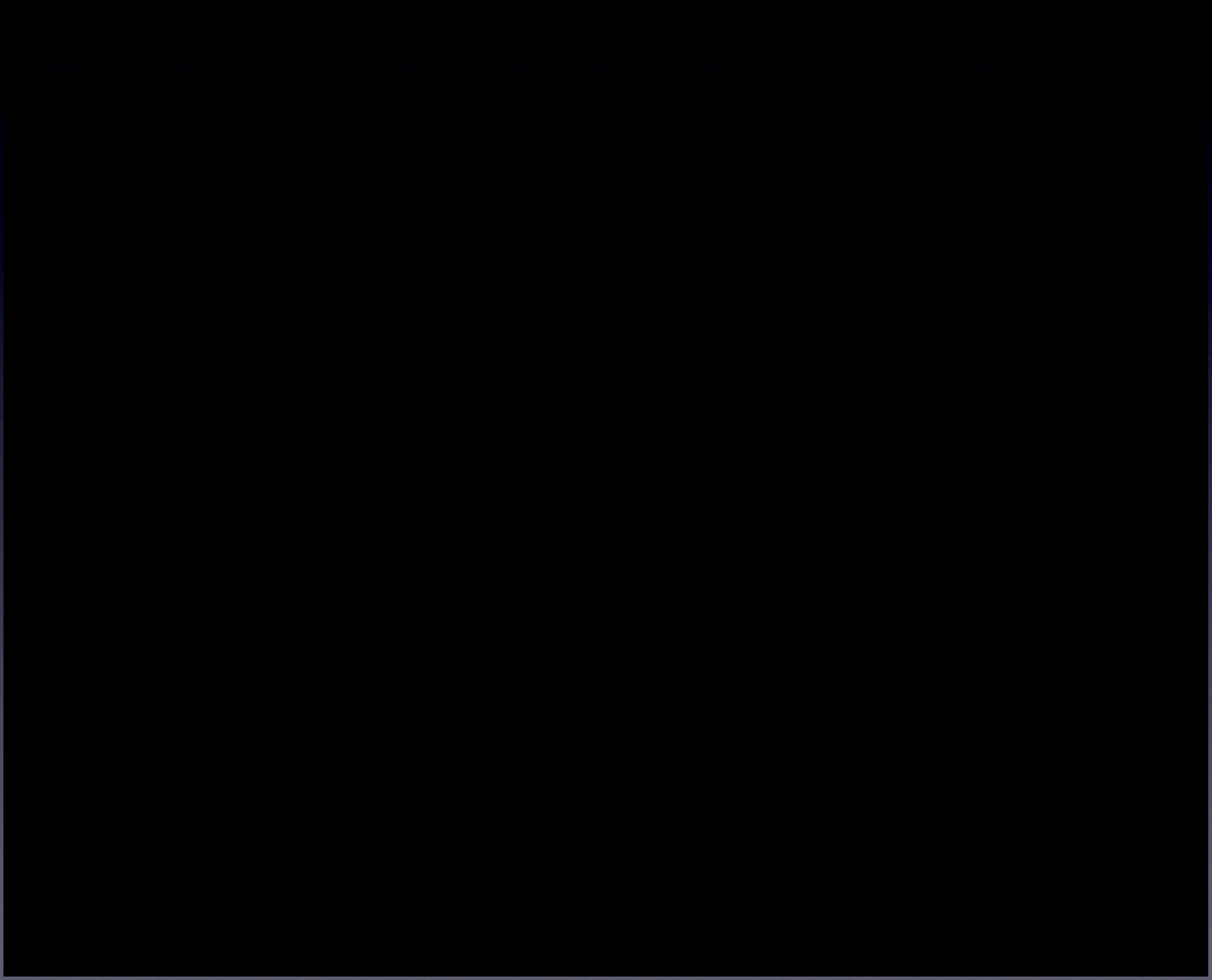
Flight Control



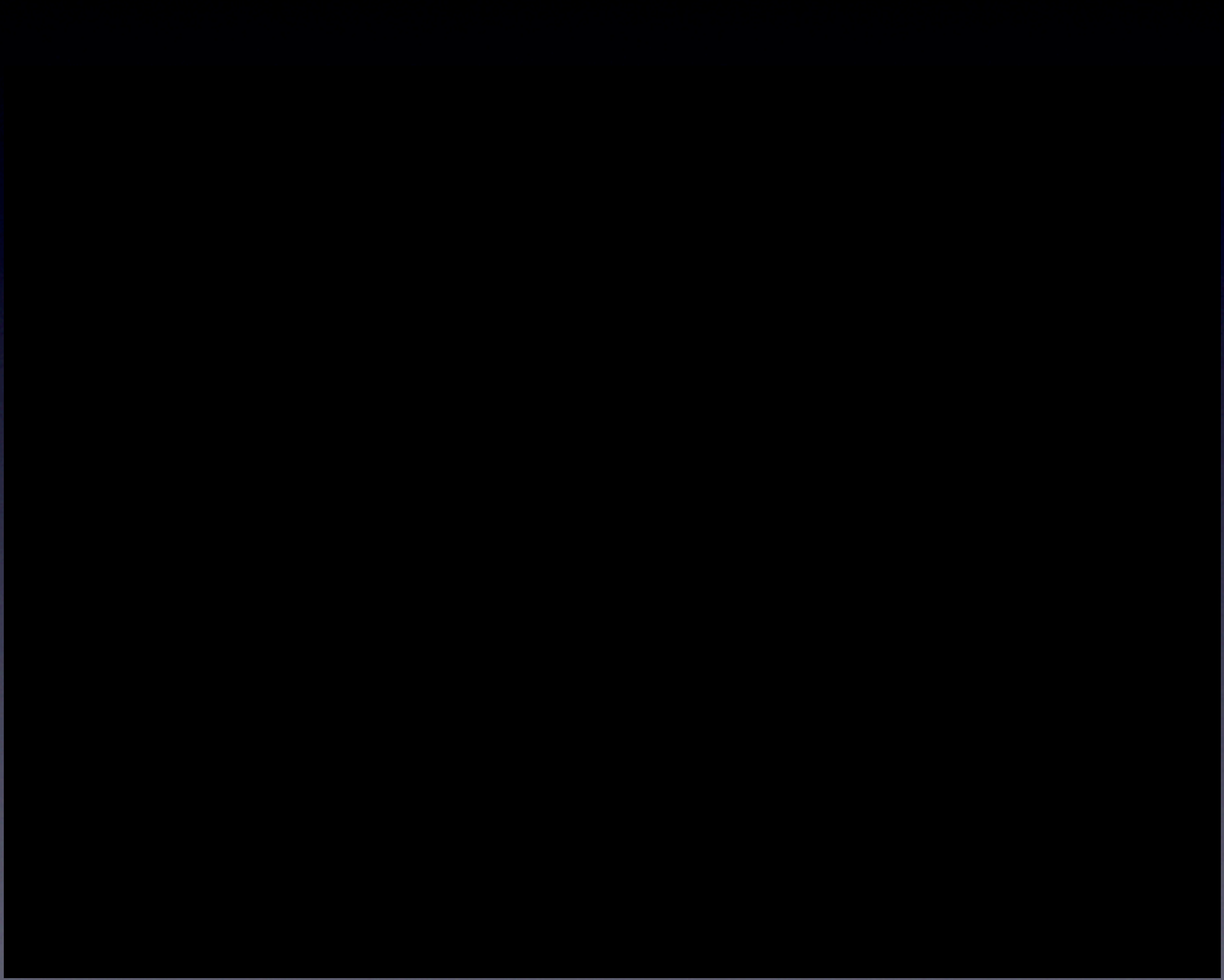
Yaw Control



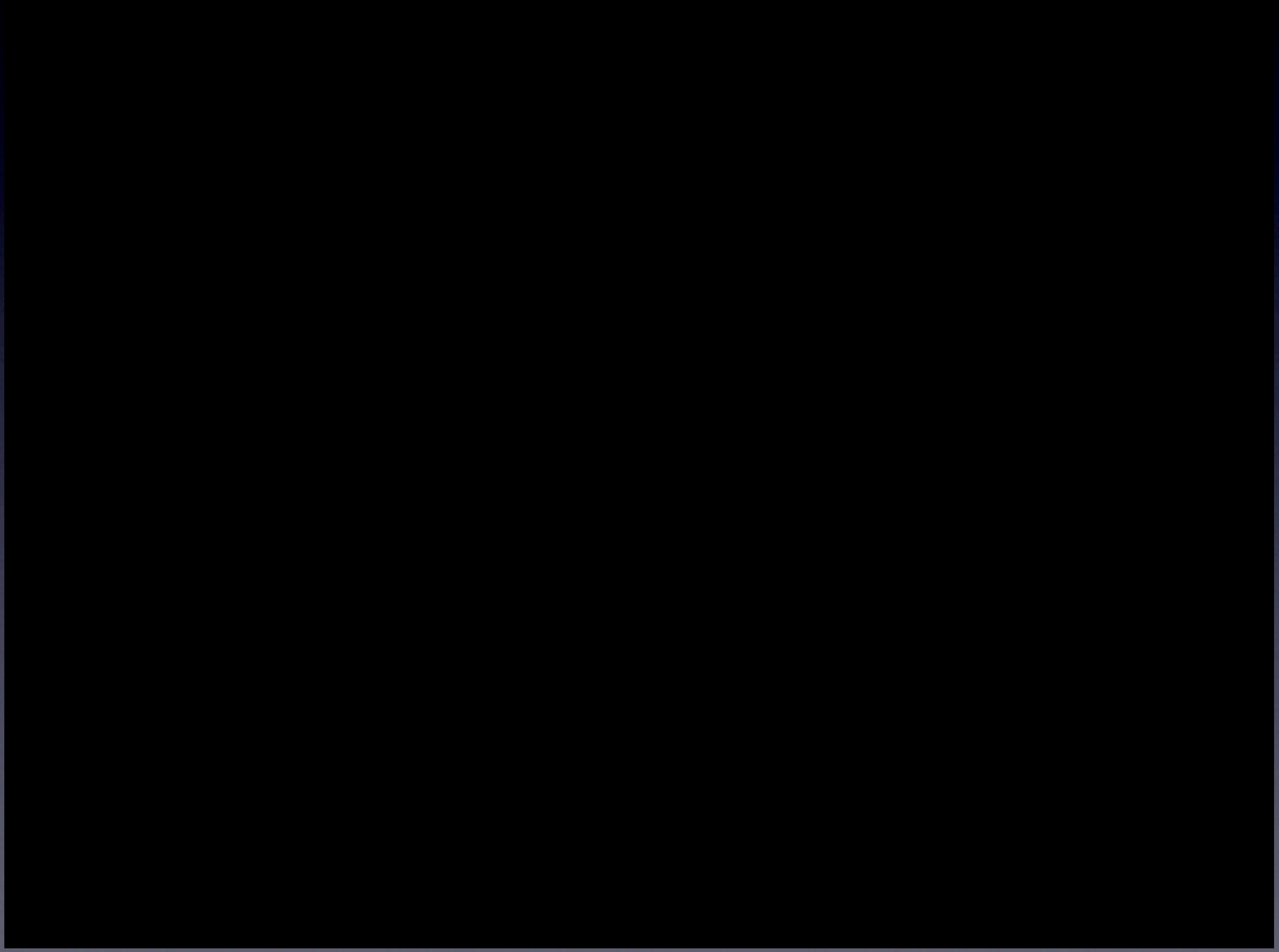
Free Flight



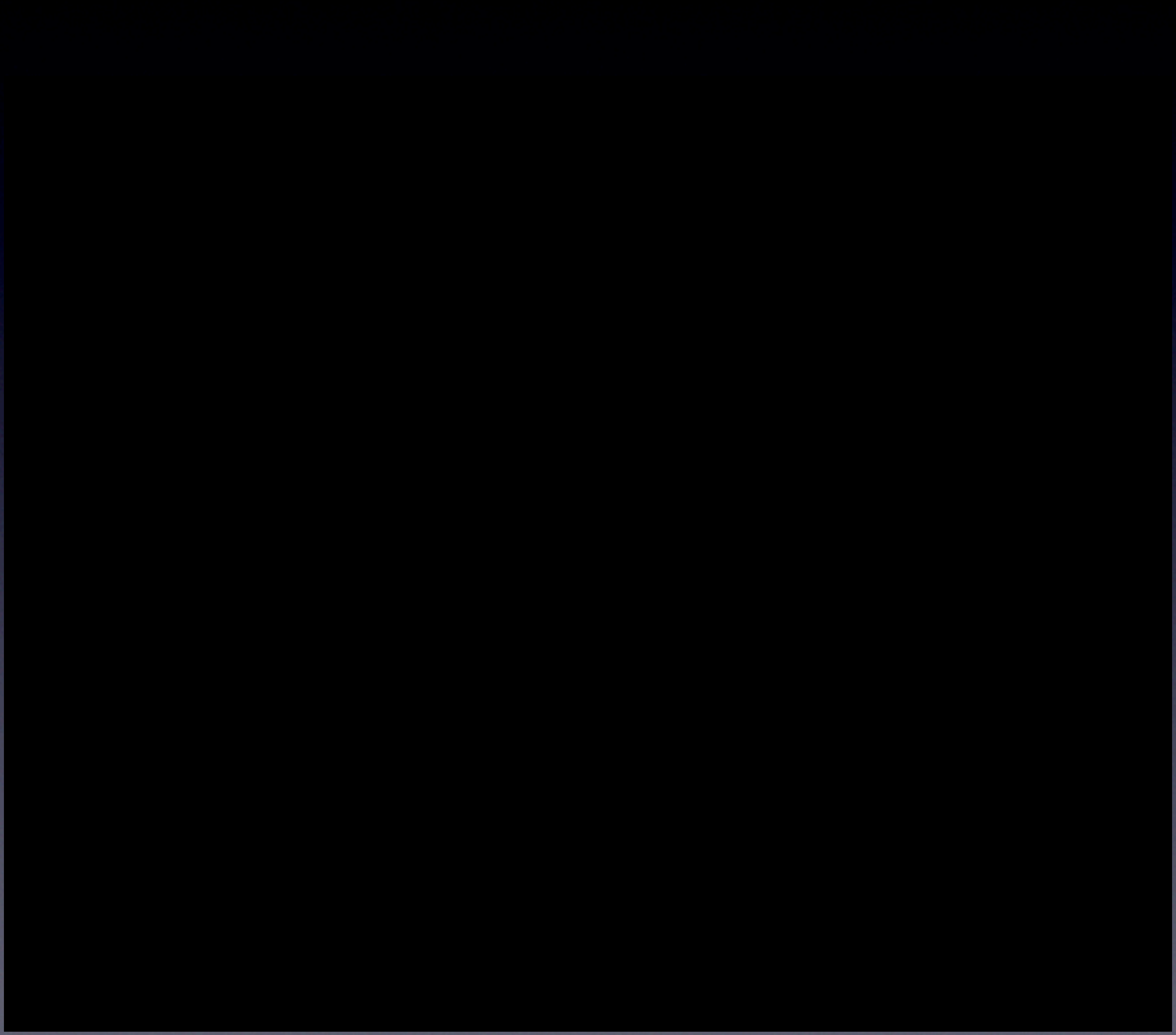
[AIAA GNC 2008]



[AIAA GNC 2008]



Fun Stuff



Time-Portable Programming

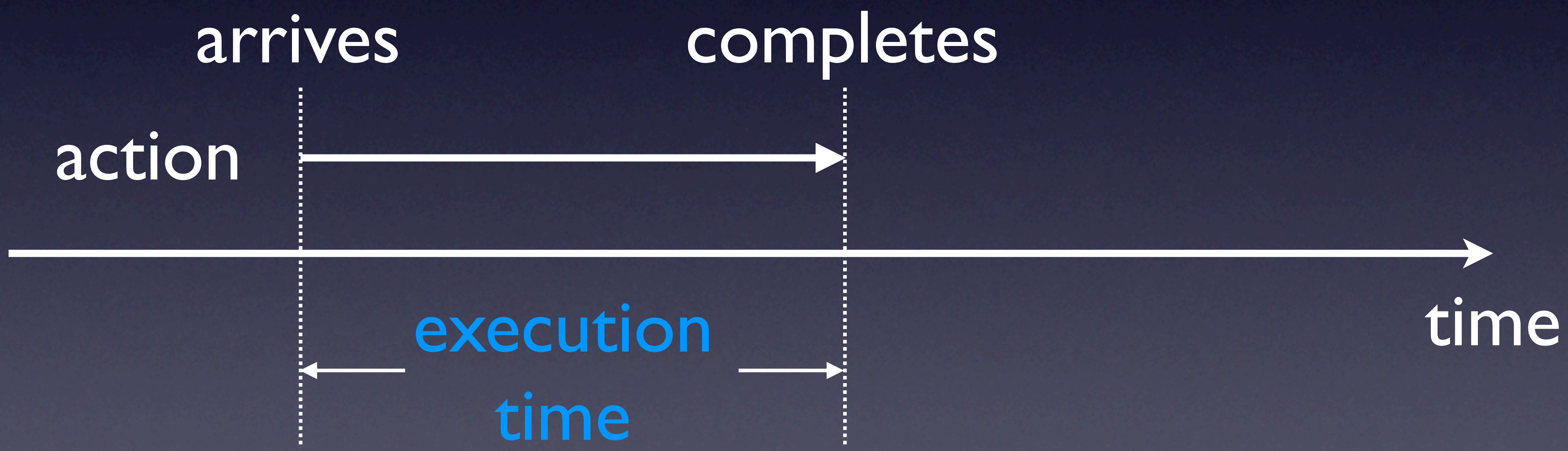
Exotasks

Tiptoe

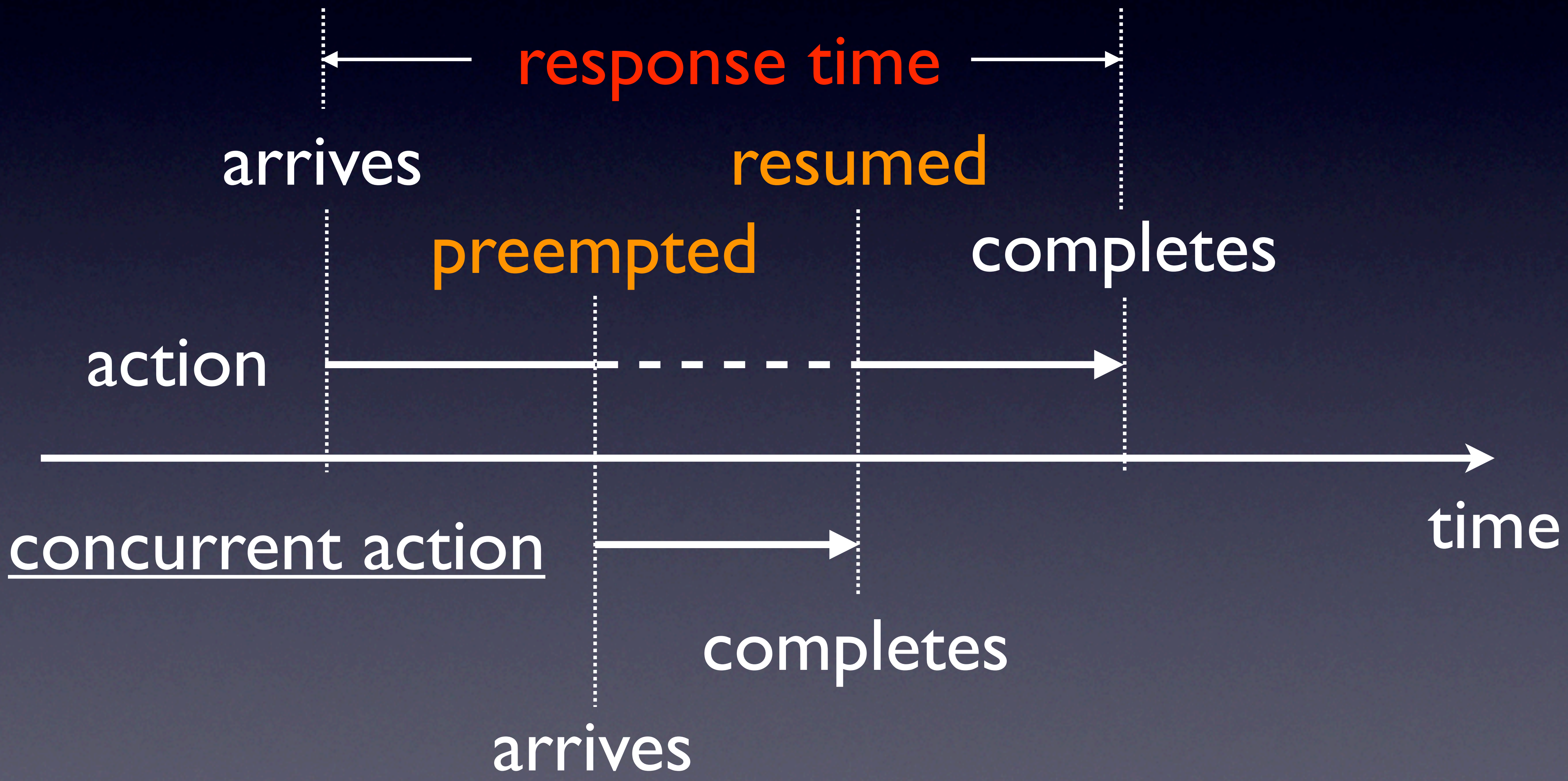
Outline

1. Time-Portable Programming
2. Exotasks (Java)
3. Tiptoe (C)

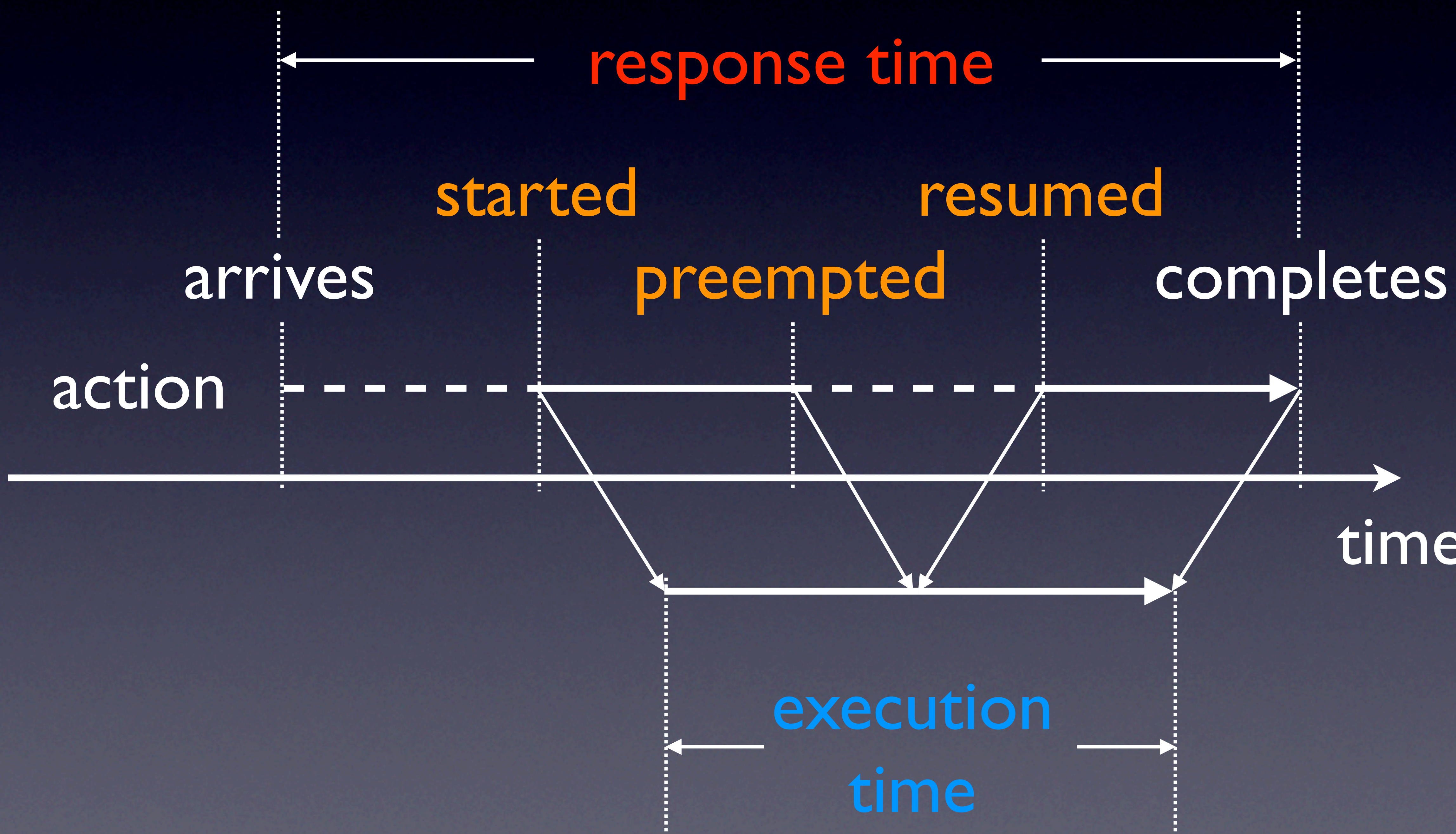
Process Action



Concurrency



Execution and Response



Time

Time

- The temporal behavior of a process action is characterized by its execution time and its response time

Time

- The temporal behavior of a process action is characterized by its execution time and its response time
- The execution time is the time it takes to execute the action in the absence of concurrent activities

Time

- The temporal behavior of a process action is characterized by its execution time and its response time
- The execution time is the time it takes to execute the action in the absence of concurrent activities
- The response time is the time it takes to execute the action in the presence of concurrent activities

Time-Portable Programming

Time-Portable Programming

- Time-portable programming specifies and implements upper AND lower bounds on response times of process actions

Time-Portable Programming

- Time-portable programming specifies and implements upper AND lower bounds on response times of process actions
- A program is time-portable if the response times of its process actions are maintained across different hardware platforms and software workloads

Time-Portable Programming

- Time-portable programming specifies and implements upper AND lower bounds on response times of process actions
- A program is time-portable if the response times of its process actions are maintained across different hardware platforms and software workloads
- The difference ε between upper and lower bounds is its "degree of time portability"

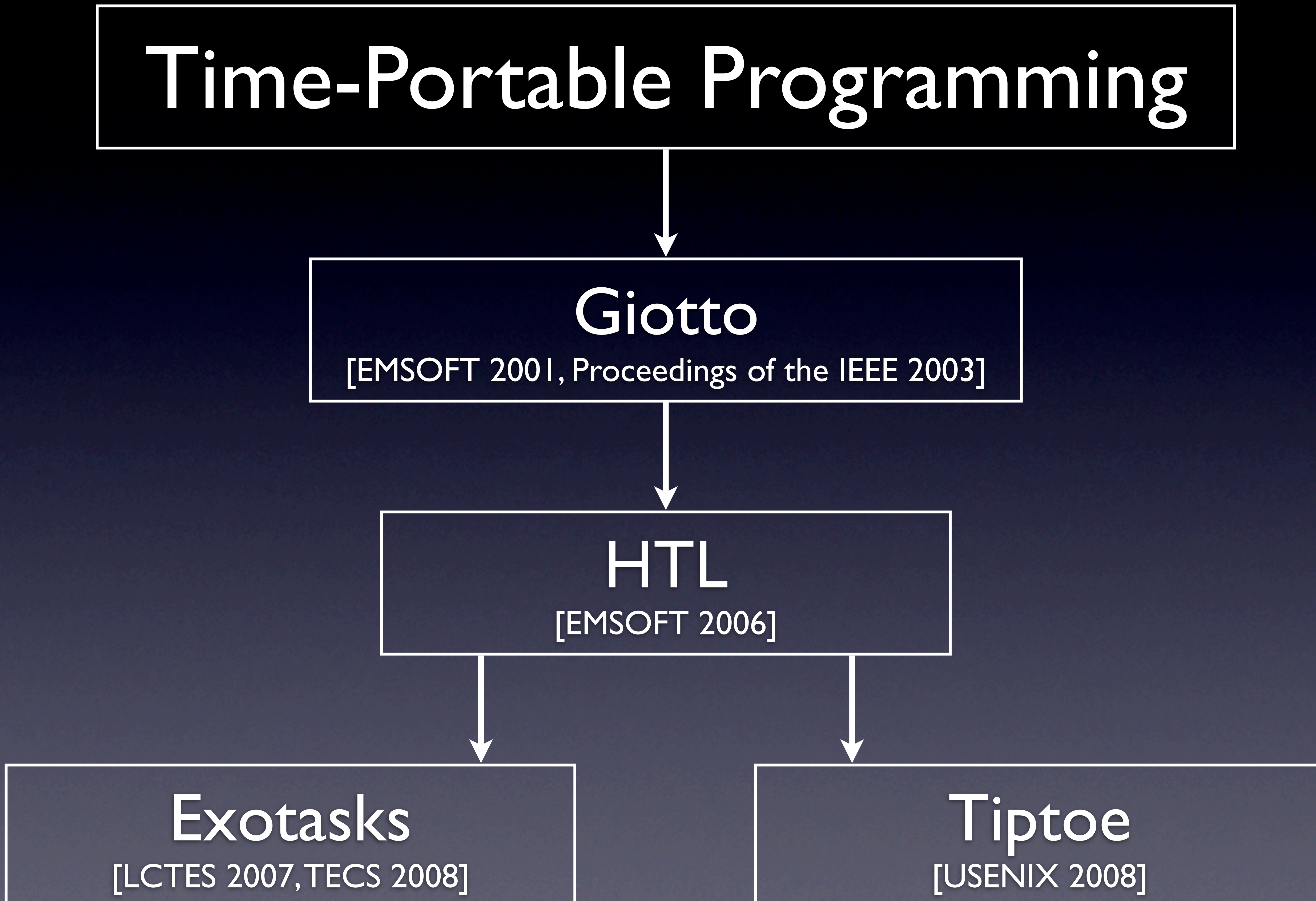
Correctness

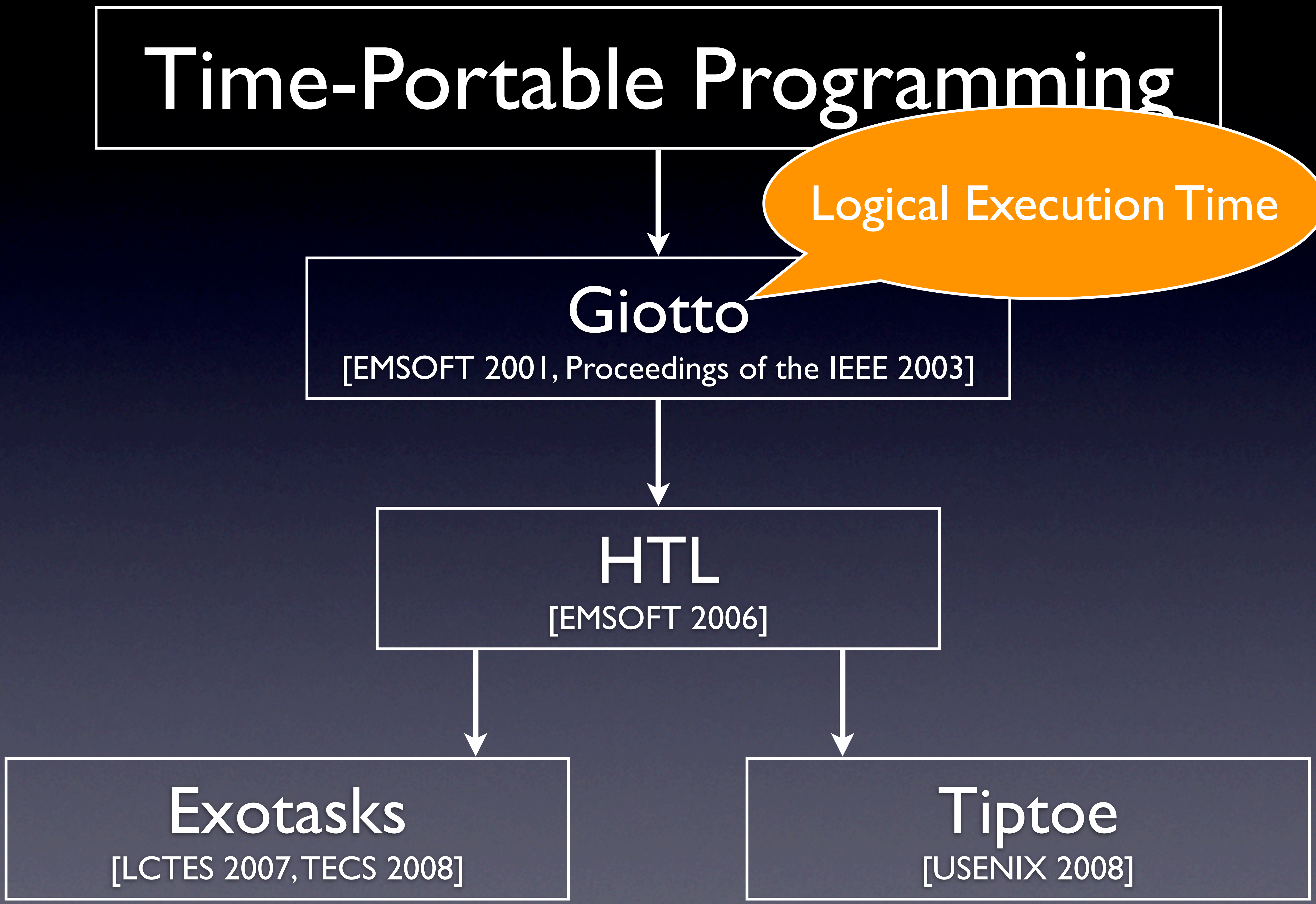
Correctness

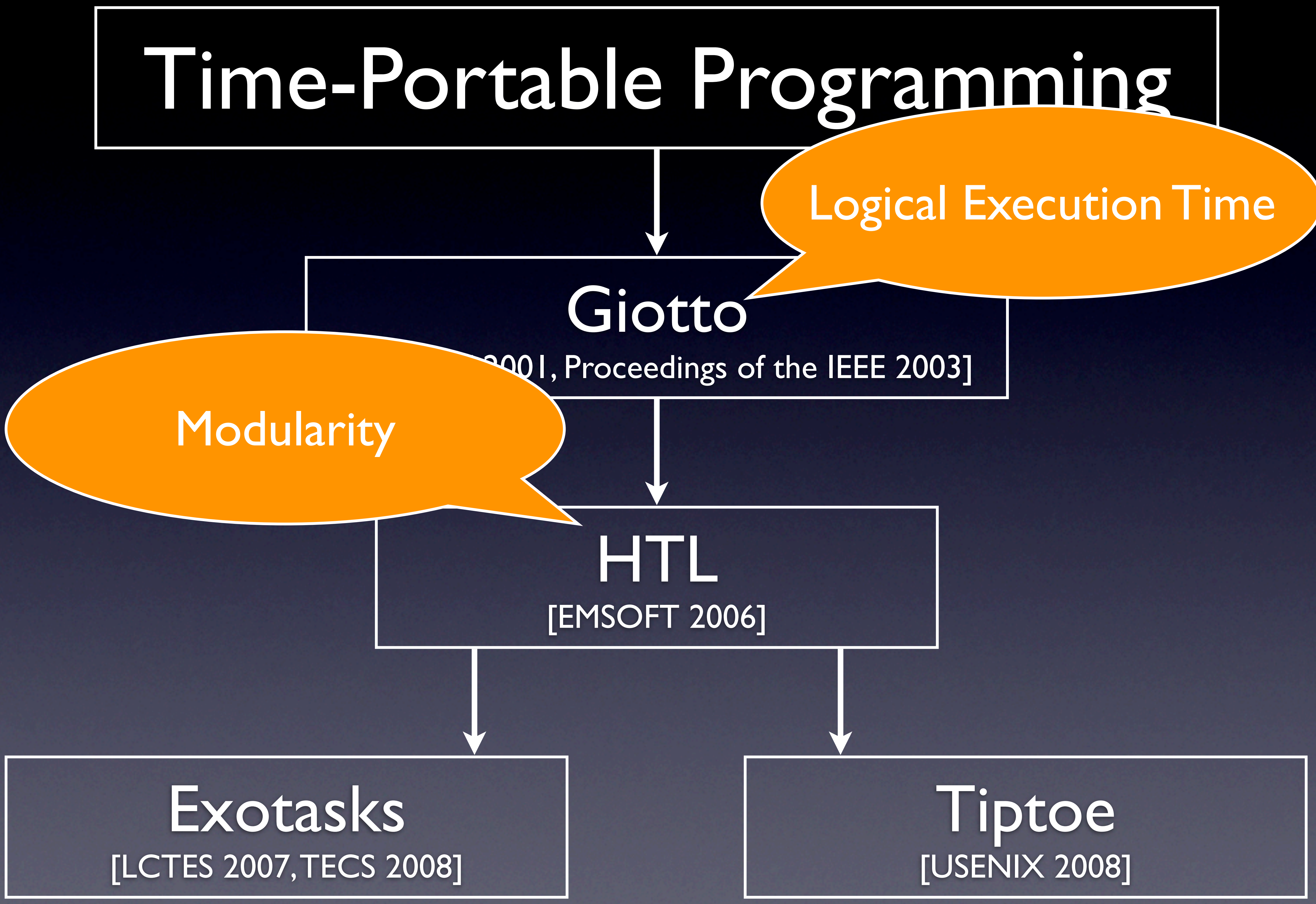
1. The execution time of a process action is determined by the process action and the executing processor.
   - Worst-case execution time (WCET) analysis

Correctness

1. The execution time of a process action is determined by the process action and the executing processor.
    - Worst-case execution time (WCET) analysis
2. The response time of a process action is determined by the entire system of processes executing on a processor.
    - Real-time scheduling theory


Time-Portable Programming
Giotto
[EMSOFT 2001, Proceedings of the IEEE 2003]
HTL
[EMSOFT 2006]
Exotasks
[LCTES 2007, TECS 2008]
Tiptoe
[USENIX 2008]


Time-Portable Programming
Logical Execution Time
Giotto
[EMSOFT 2001, Proceedings of the IEEE 2003]
HTL
[EMSOFT 2006]
Exotasks
[LCTES 2007, TECS 2008]
Tiptoe
[USENIX 2008]


Time-Portable Programming
Logical Execution Time
Giotto
2001, Proceedings of the IEEE 2003]
Modularity
HTL
[EMSOFT 2006]
Exotasks
[LCTES 2007, TECS 2008]
Tiptoe
[USENIX 2008]

Time-Portable Programming

Logical Execution Time

Giotto

2001, Proceedings of the IEEE 2003]

Modularity

HTL

Java!

Exotasks

[LCTES 2007, TECS 2008]

Tiptoe

[USENIX 2008]

Time-Portable Programming

Logical Execution Time

Giotto

001, Proceedings of the IEEE 2003]

Modularity

HTL

C!

Exotasks

[LCTES 2007, TECS 2008]

Tiptoe

[USENIX 2008]

Outline

1. Time-Portable Programming
2. Exotasks (Java)
3. Tiptoe (C)

Exotask Team#

- J. Auerbach, D.F. Bacon, V.T. Rajan (IBM Research)
- Daniel Iercan (TU Timisoara, Romania)

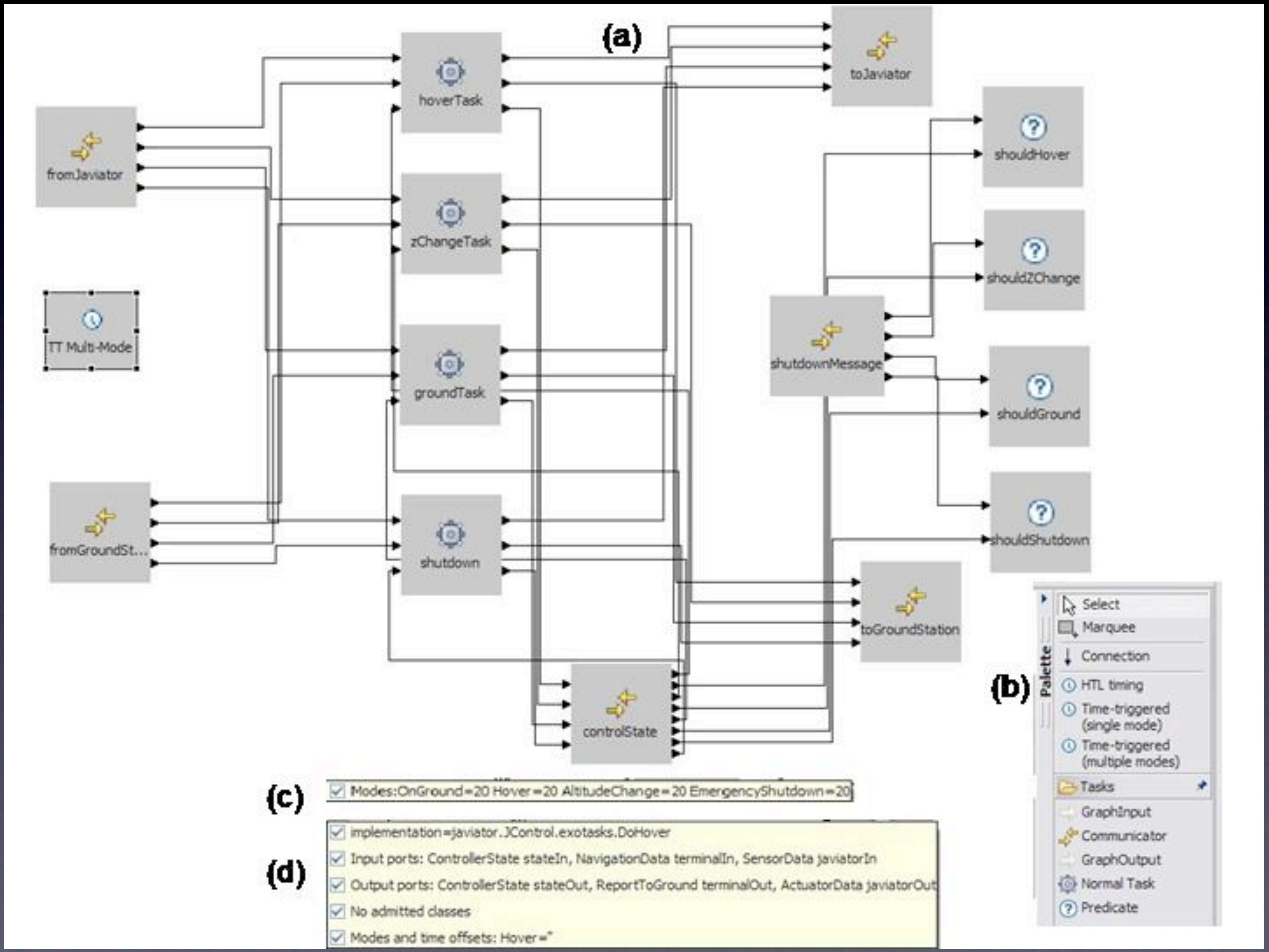
- Silviu Craciunas* (Univ. of Salzburg, Austria)
- Harald Röck (Univ. of Salzburg, Austria)
- Rainer Trummer (Univ. of Salzburg, Austria)

#Supported by a 2007 IBM Faculty Award and the EU ArtistDesign Network of Excellence on Embedded Systems Design
*Supported by Austrian Science Fund Project P18913-N15

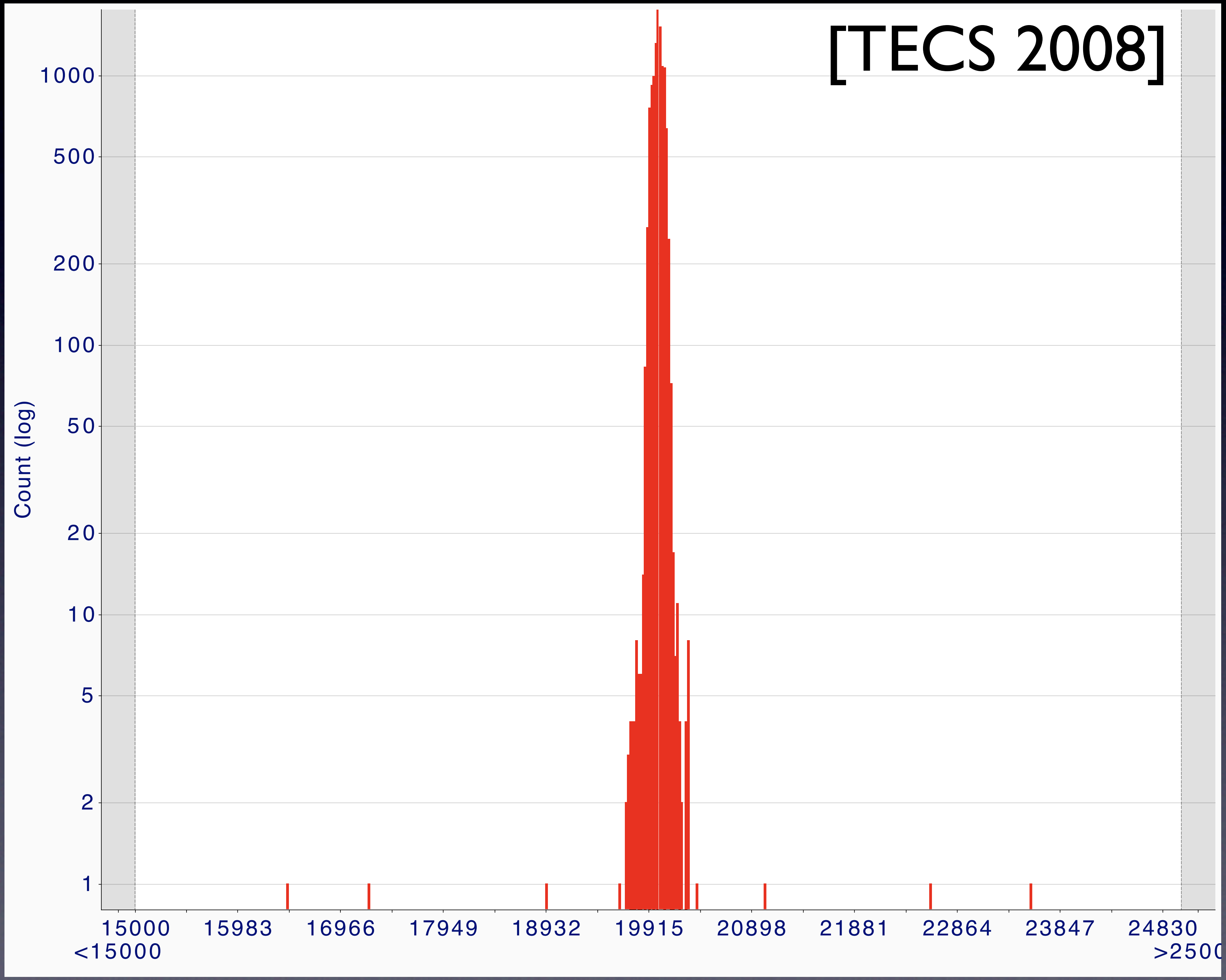
Exotasks

- Alternative to Java threads
- Single-threaded code: <u>validated</u> Java subset
- Isolated in space: private heaps, <u>individual</u> GC
- Communicate by <u>message-passing</u> Java objects
- Isolated in time: HTL semantics
- Other semantics are possible: scheduler plugins

Eclipse Plugin



Performance Histogram



@256KB/s

~1ms

Outline

1. Time-Portable Programming
2. Exotasks (Java)
3. Tiptoe (C)

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)

#Supported by a 2007 IBM Faculty Award and the EU ArtistDesign Network of Excellence on Embedded Systems Design
*Supported by Austrian Science Fund Project P18913-N15

Example Process

```
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);
```

Example Process

Workload Parameter

```
{
      ...ames = 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

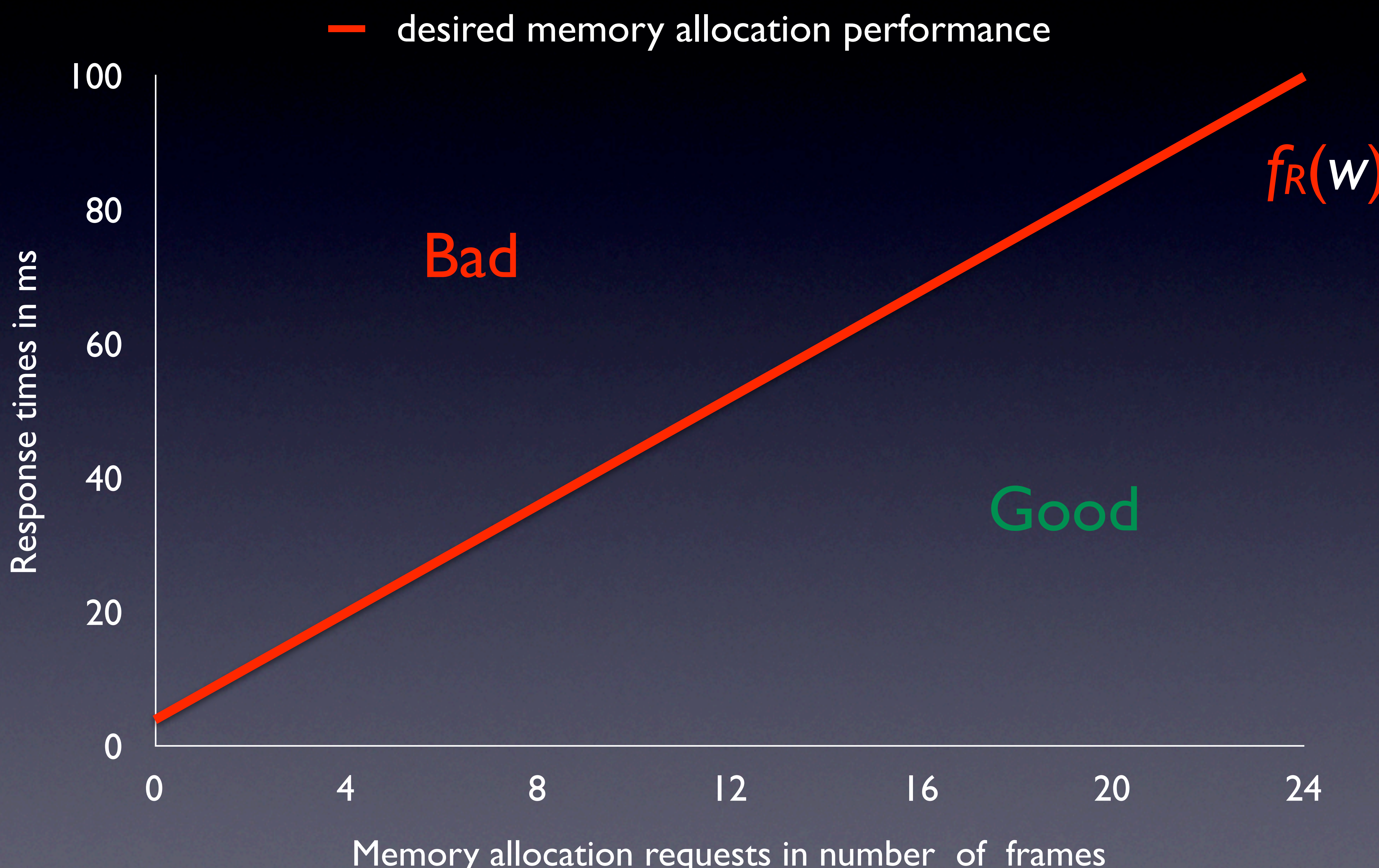
- Process actions are characterized by their execution time and response time in terms of their optional workload parameters

[USENIX 2008]

- malloc(n) takes O(1)
- free(n) takes O(1) (or O(n) if compacting)
- access takes one indirection

- memory fragmentation is bounded and predictable in constant time

Response-Time Function



Throughput & Latency

f_R(1 frame) = 8ms but only 125fps

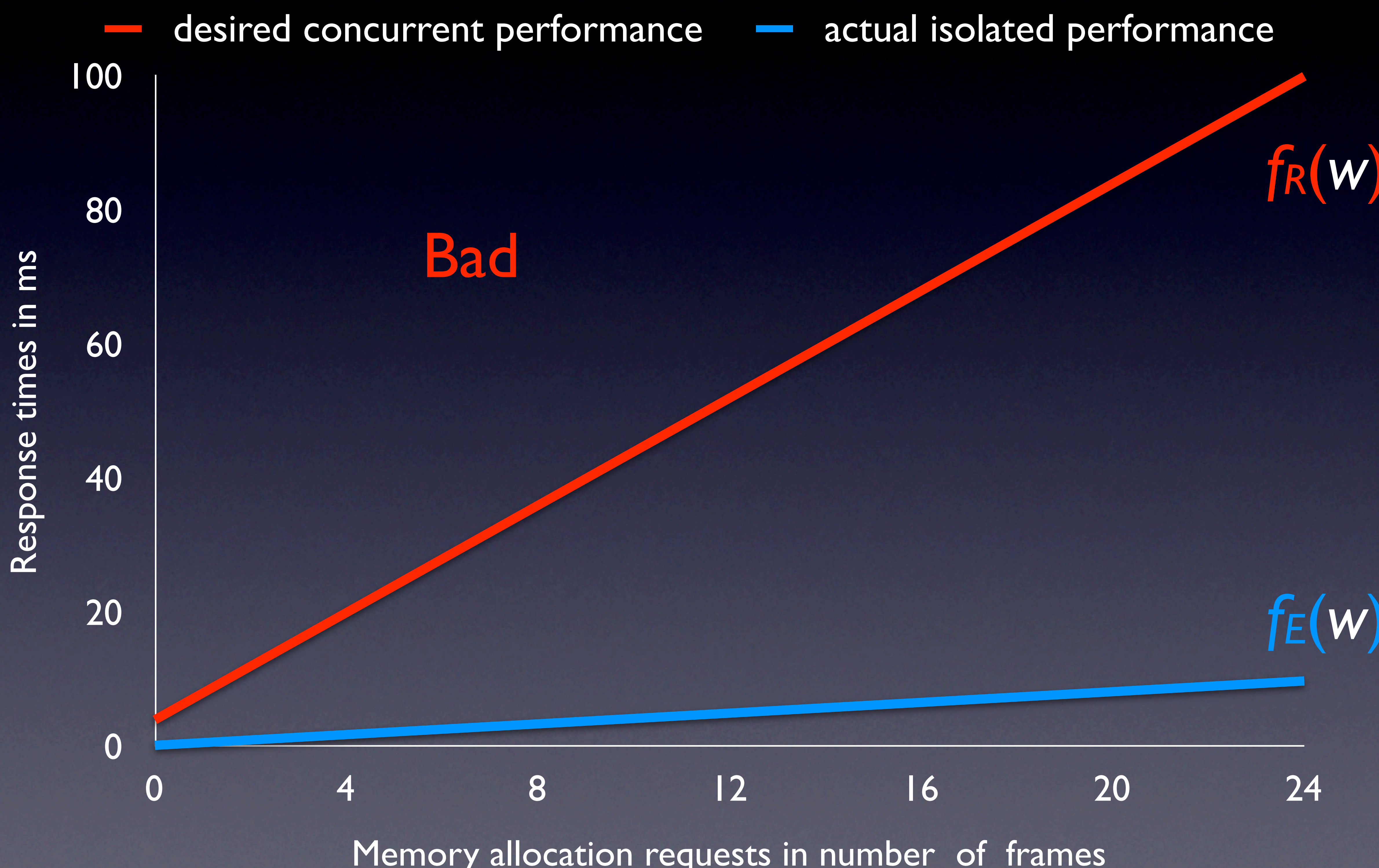
...

f_R(4 frames) = 20ms yields 200fps

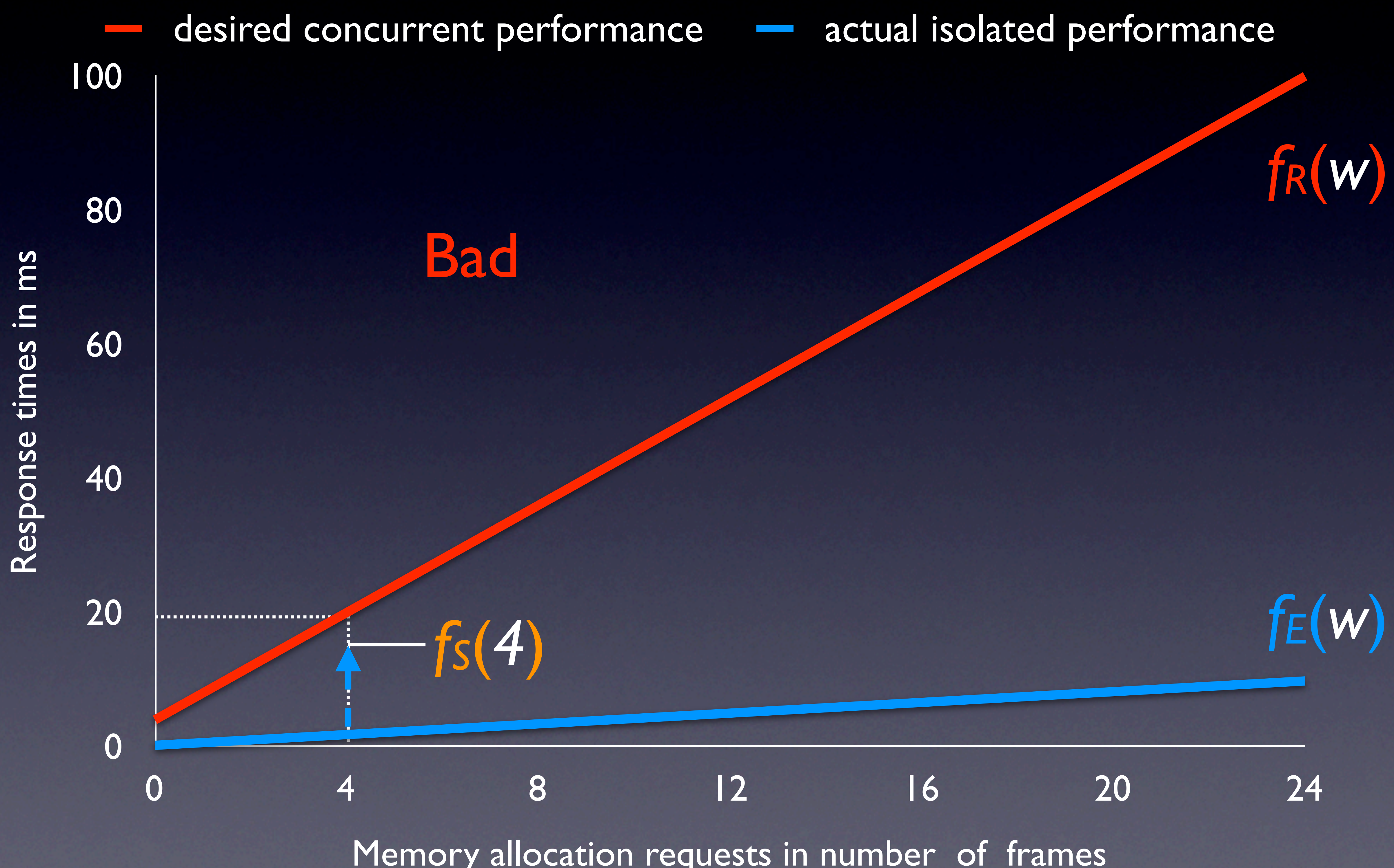
...

f_R(24 frames) = 100ms yet 240fps

Execution-Time Function



Scheduled Response Time



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

and

$$\forall w.\ f_R(w) - \varepsilon \leq f_S(w)\ ?$$

with ε representing the "degree of time portability"

Scheduling and Admission

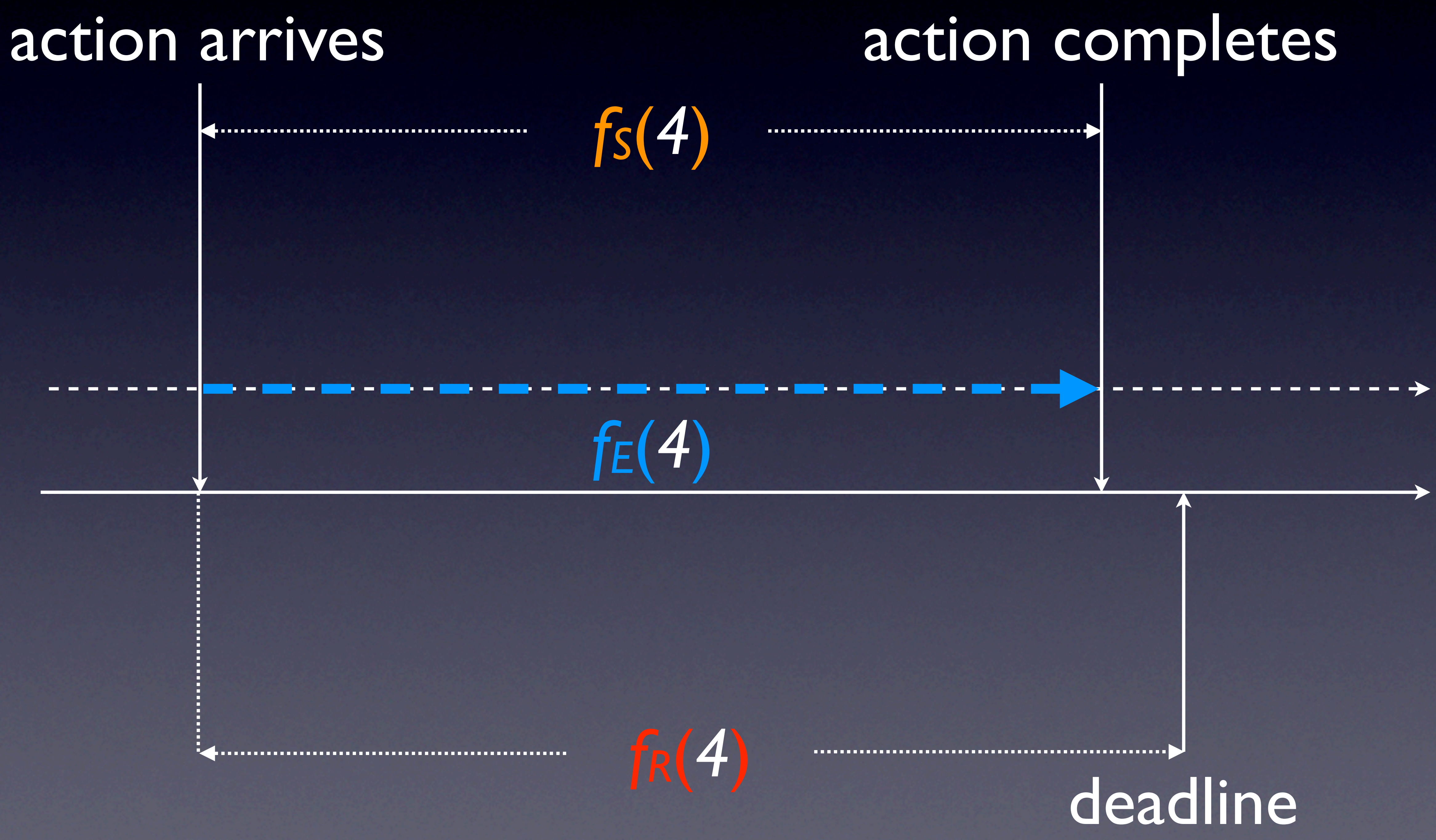
Scheduling and Admission

- Process scheduling:
  - How do we efficiently schedule processes on the level of individual process actions?

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?

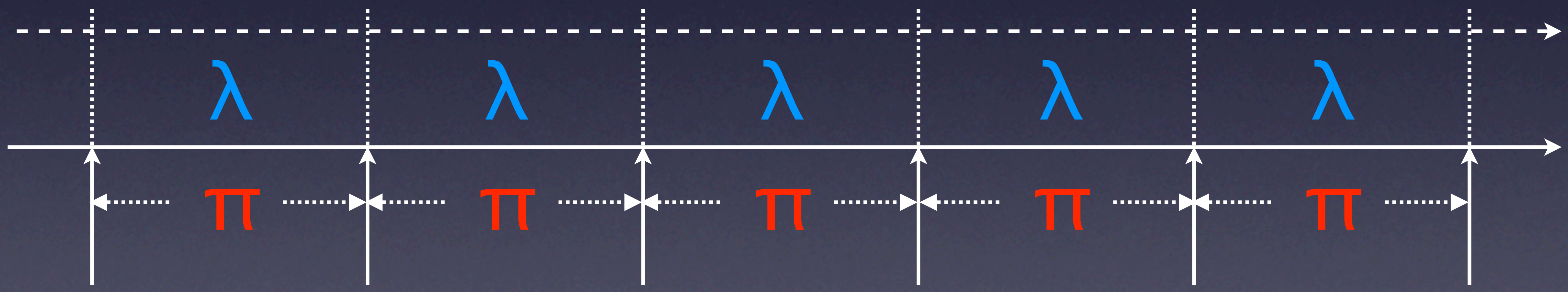


Virtual Periodic Resource

limit: λ

period: π

utilization: λ / π



Tiptoe Process Model

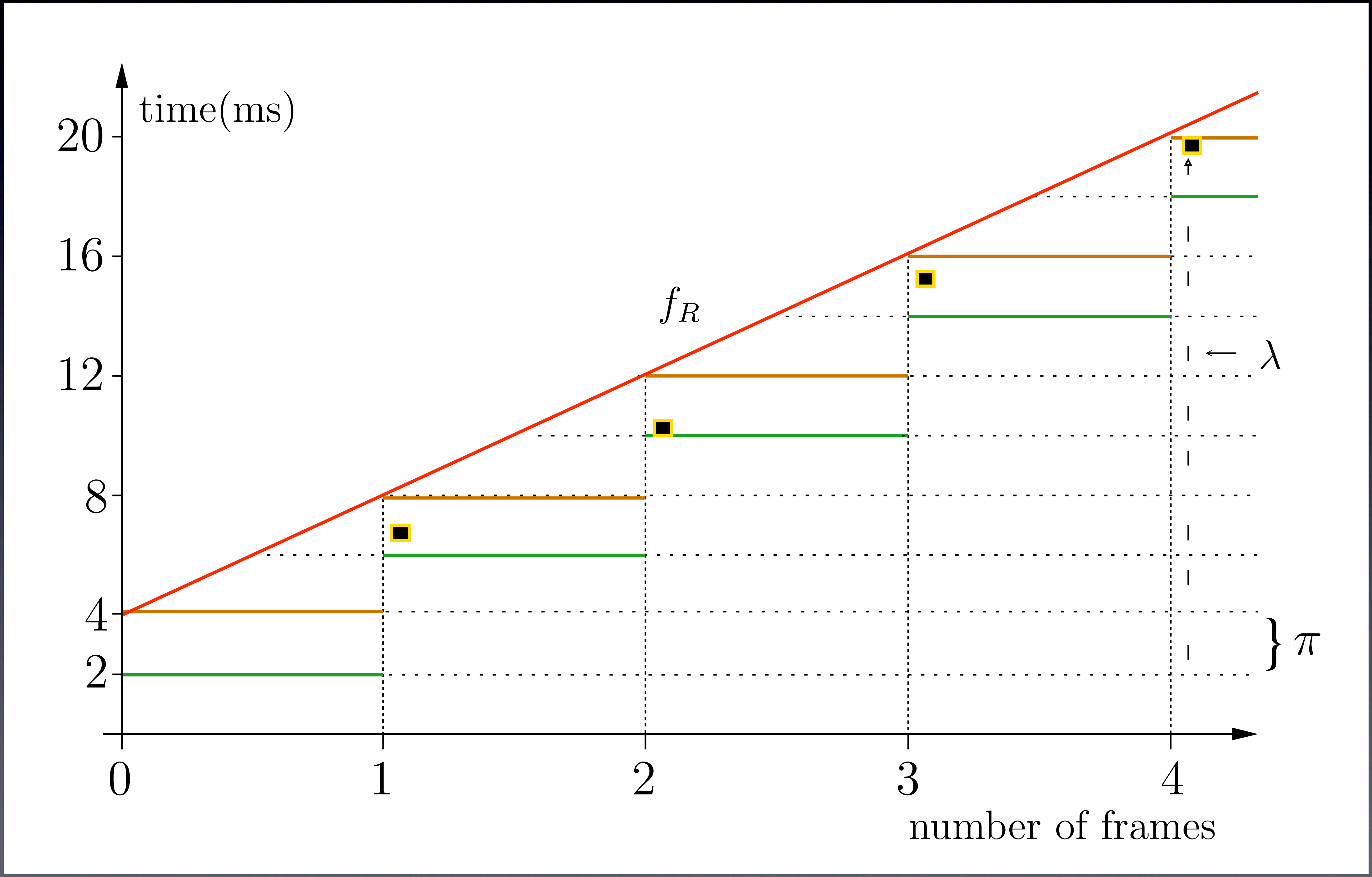
Tiptoe Process Model

- Each Tiptoe process declares a finite set of virtual periodic resources

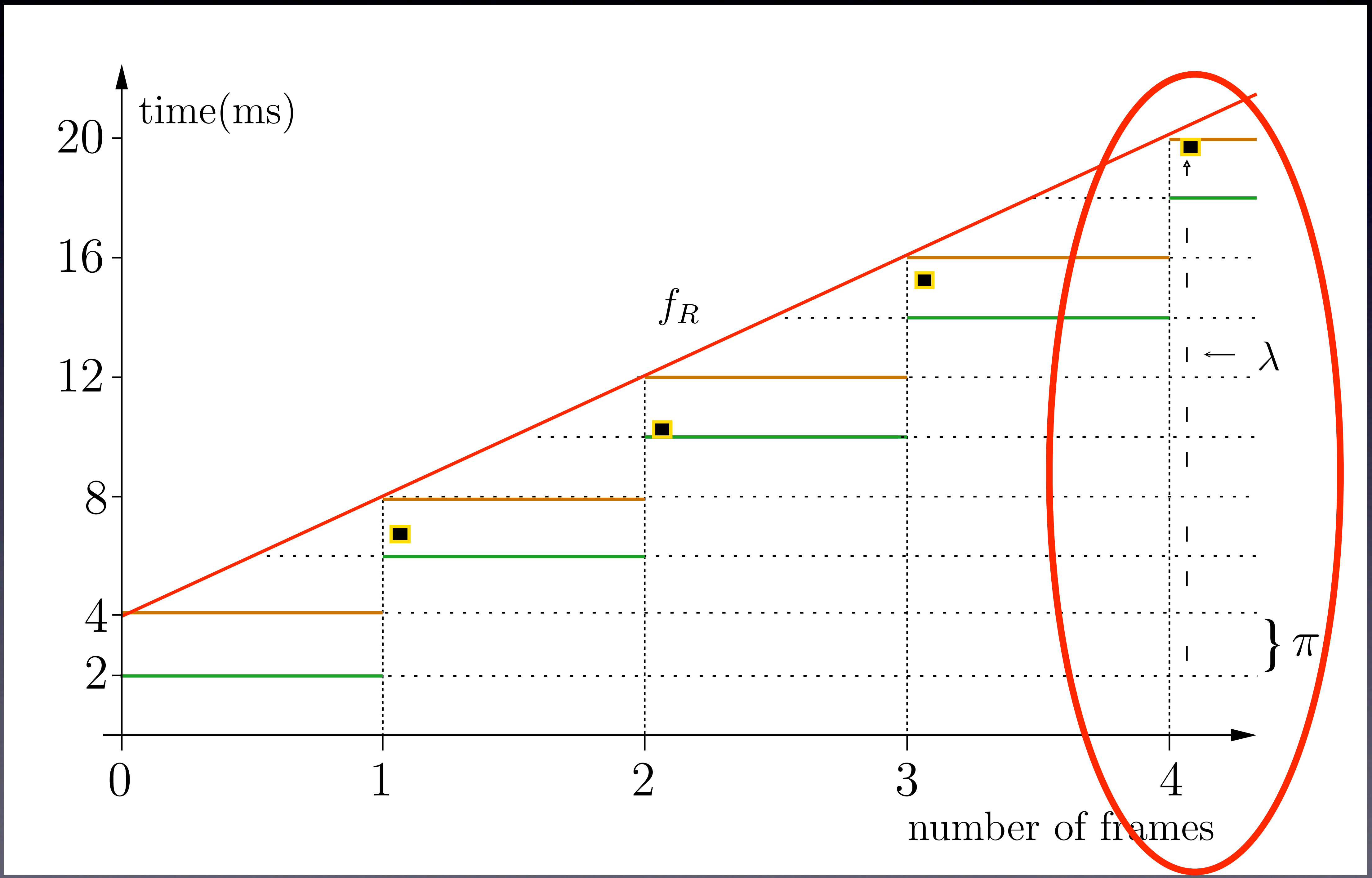
Tiptoe Process Model

- 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

f_R(4 frames) = 20ms
λ = 200µs; π = 2ms



f_R(4 frames) = 20ms
λ = 200µs; π = 2ms



The smaller the π
the smaller the ε may be,
that is, the higher the
"degree of time portability"
but also
the higher the
scheduling overhead

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

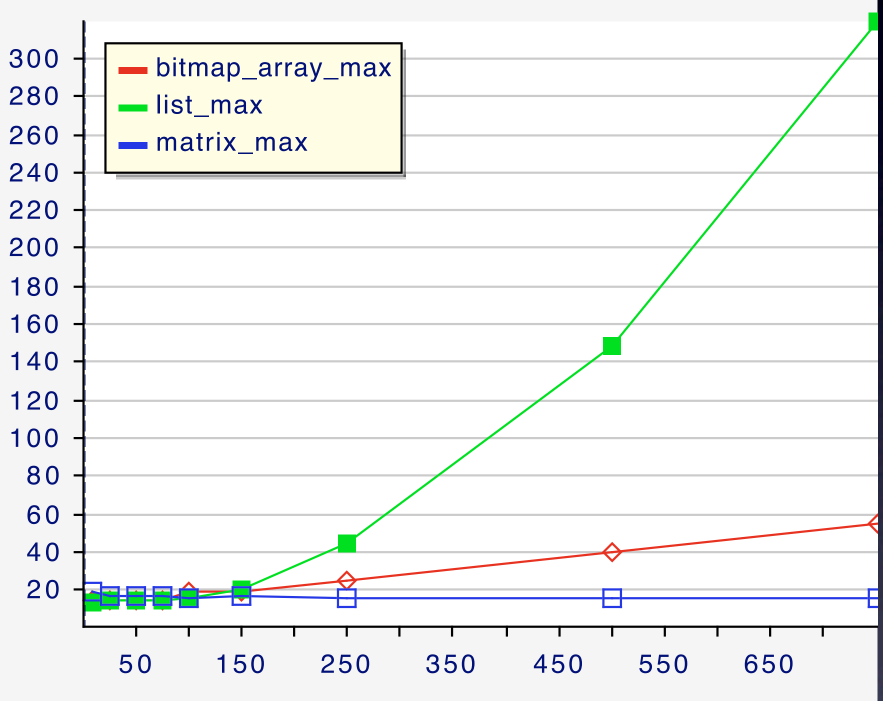
Time and Space

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

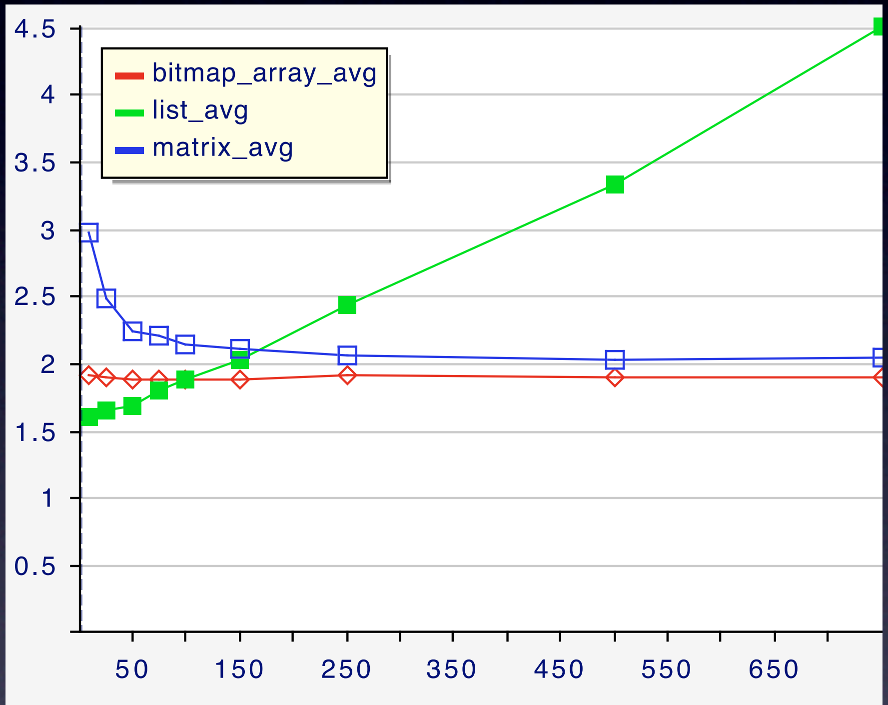
	list	array	matrix
time	$O(n^2)$	$O(\log(t) + n \cdot \log(t))$	$\Theta(t)$
space	$\Theta(n)$	$\Theta(t + n)$	$\Theta(t^2 + n)$

n: number of processes   t: number of time instants

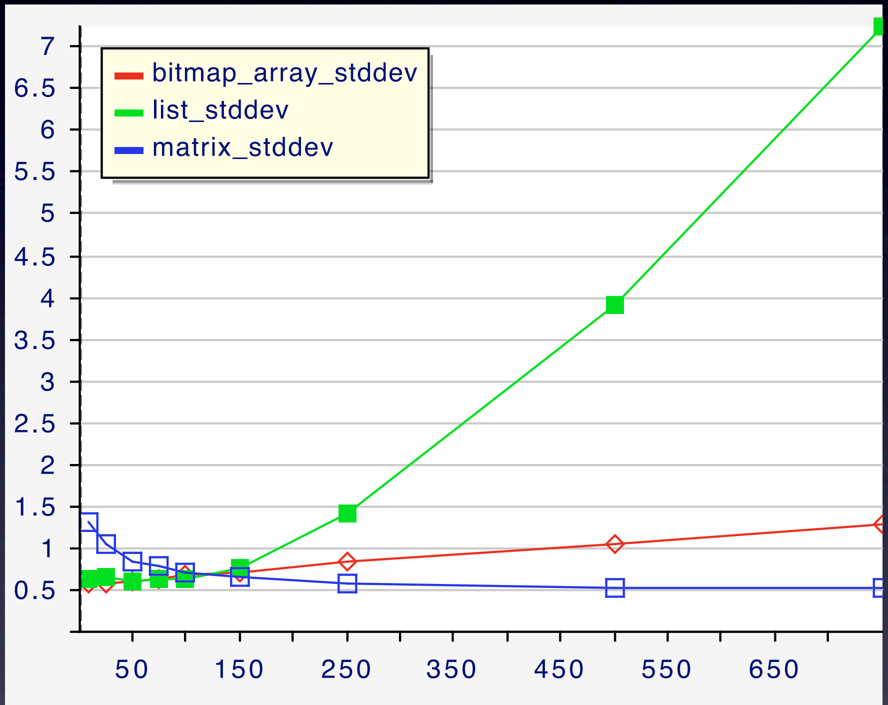
Scheduler Overhead



Max

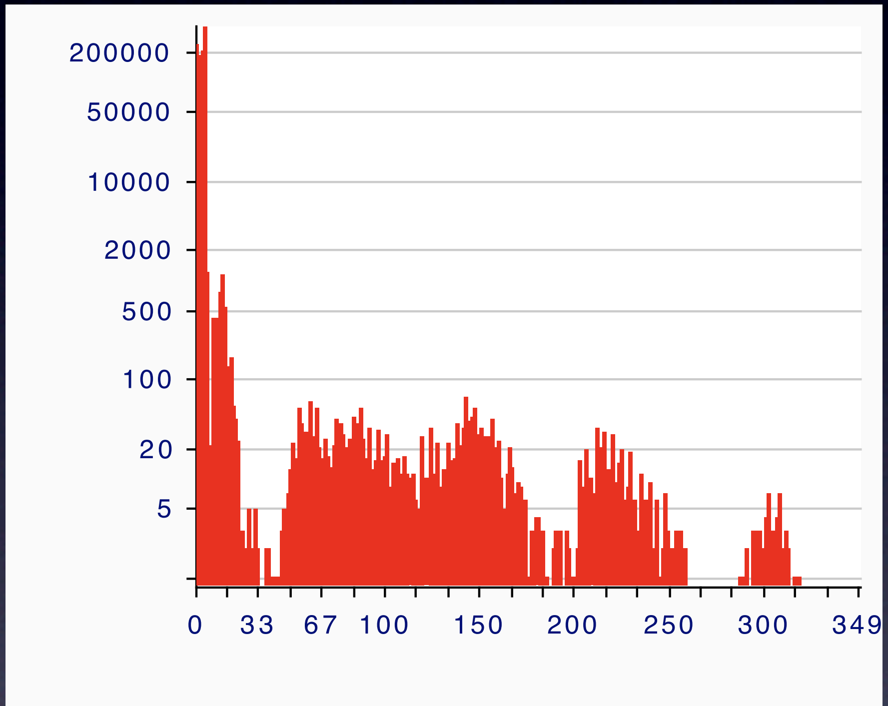


Average

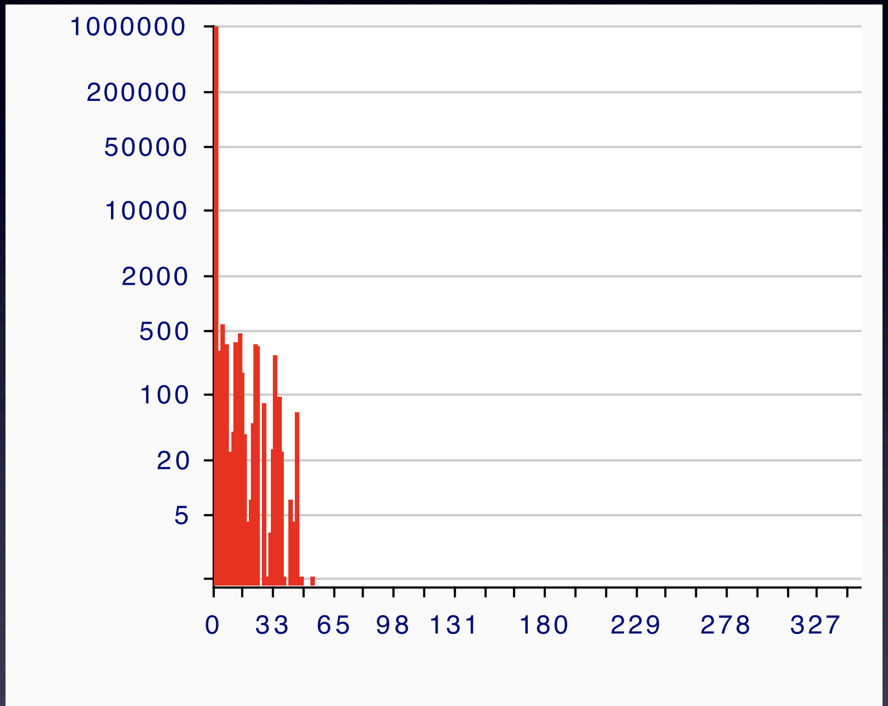


Jitter

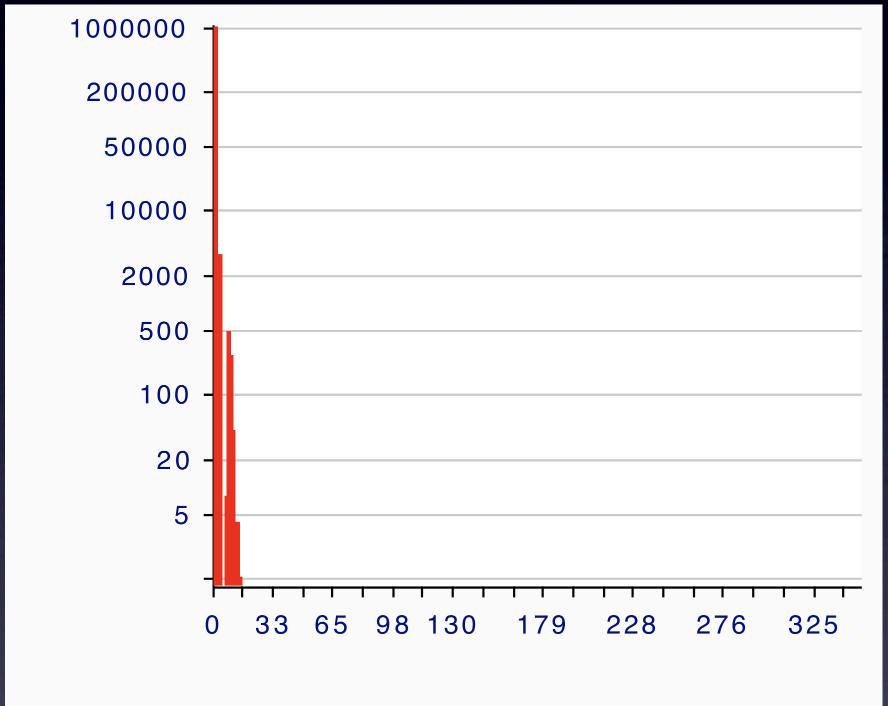
Execution Time Histograms



List

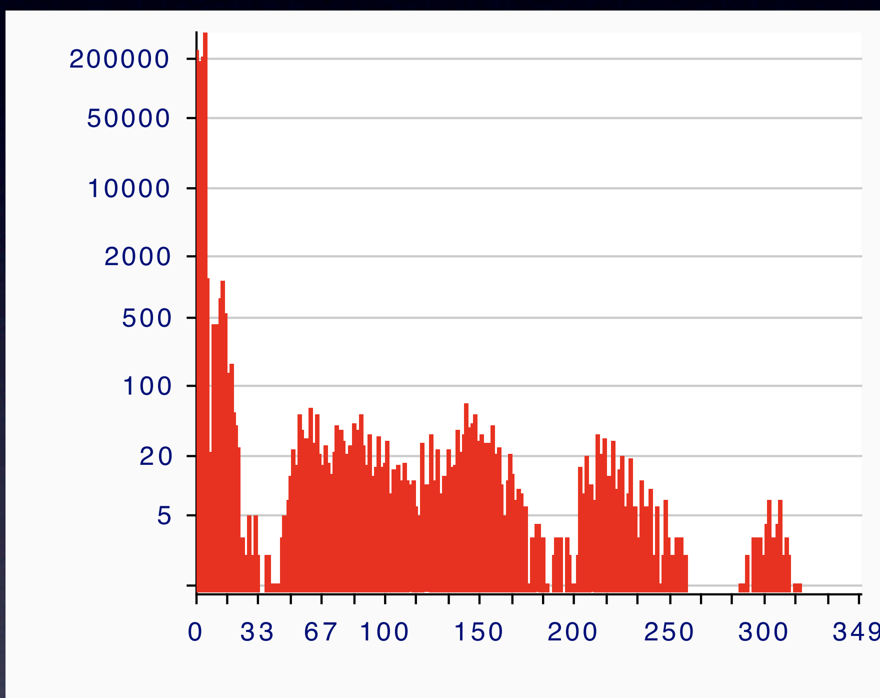


Array

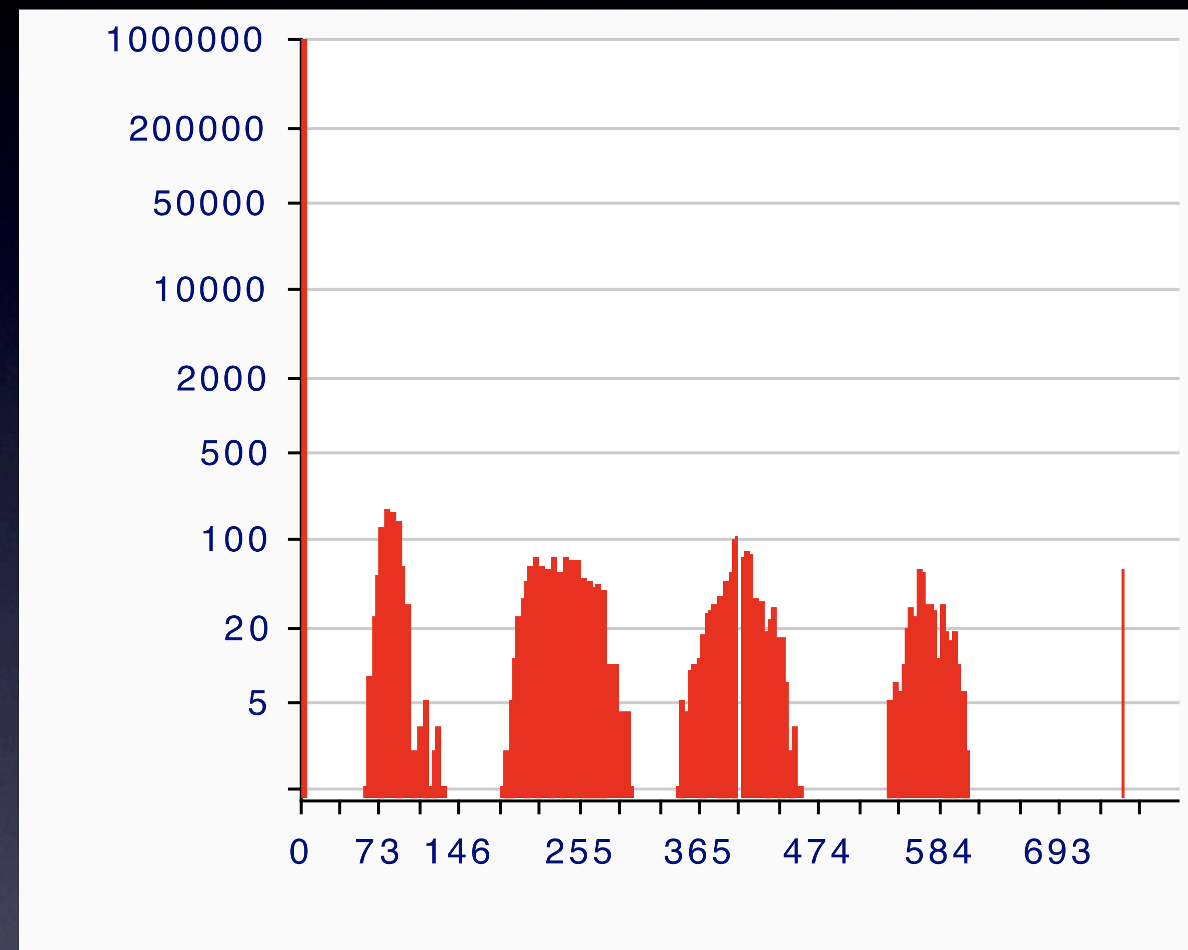


Matrix

Process Release Dominates

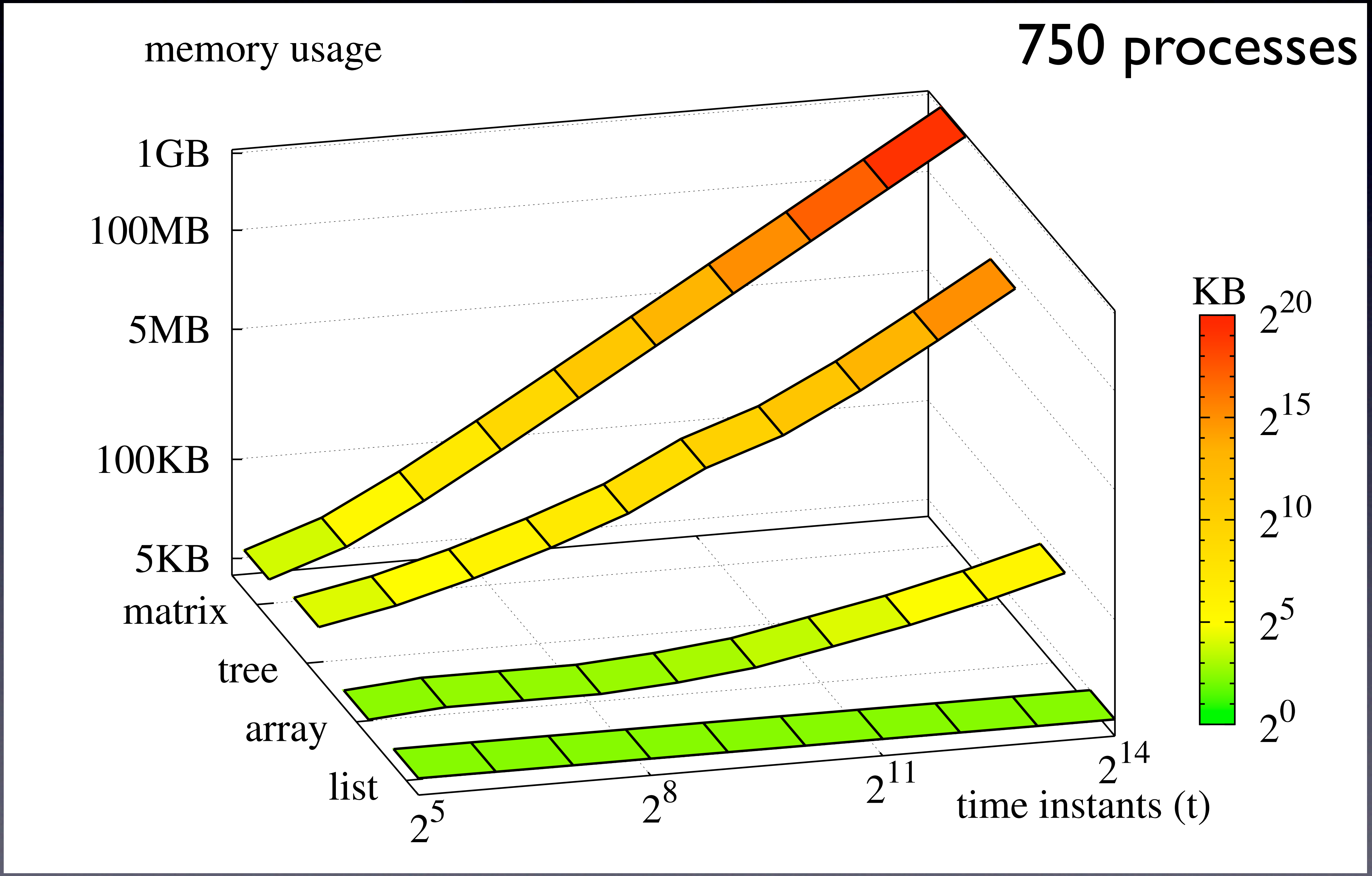


List



Releases per Instant

Memory Overhead



Current/Future Work

- Concurrent memory management
- Process management
- I/O subsystem


Thank you