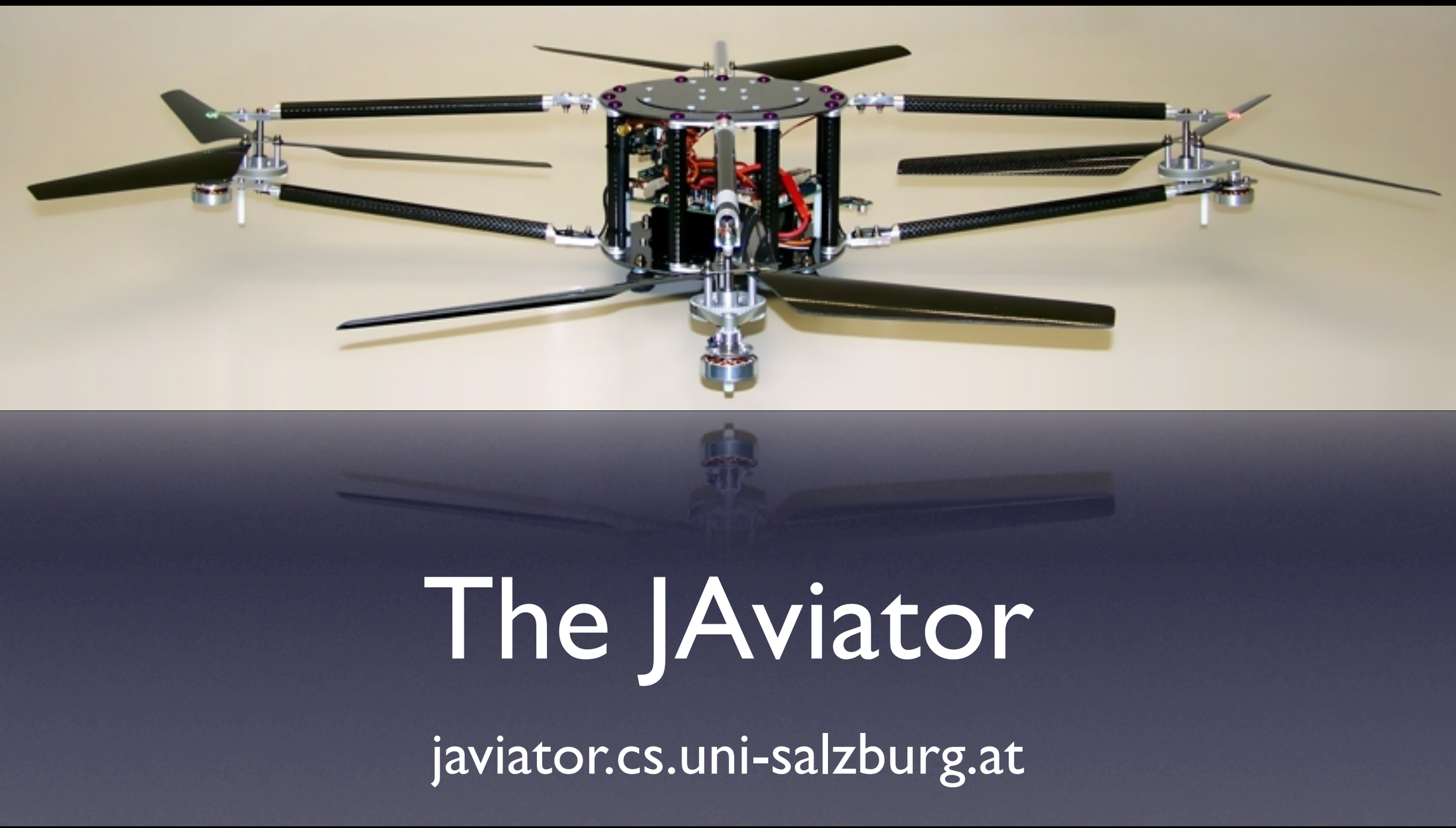
Time-Portable Programming the JAviator in Tiptoe OS

Christoph Kirsch
Universität Salzburg

UC Riverside
October 2008



The JAviator

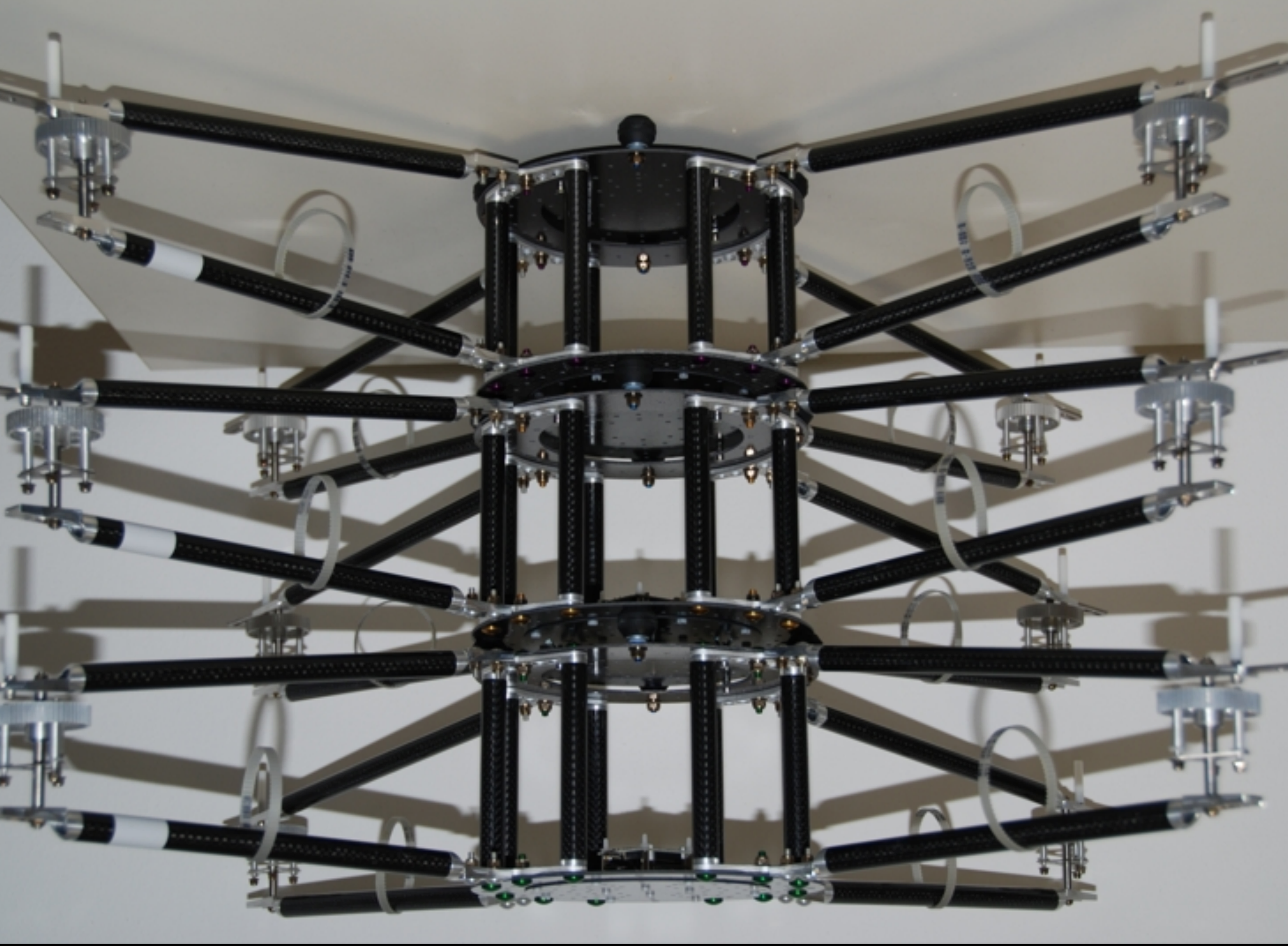
javiator.cs.uni-salzburg.at

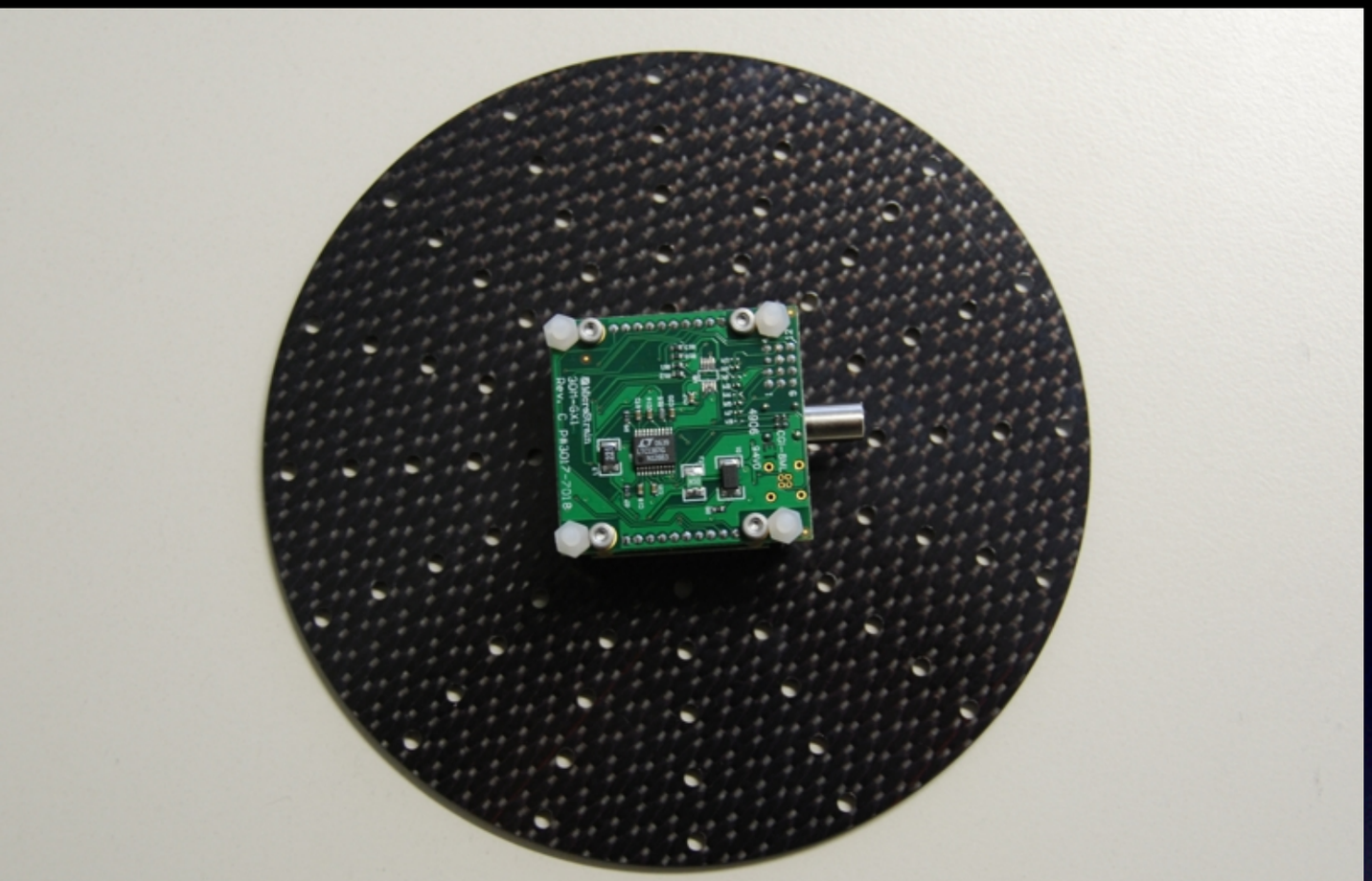
javiator.cs.uni-salzburg.at[#]

- 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

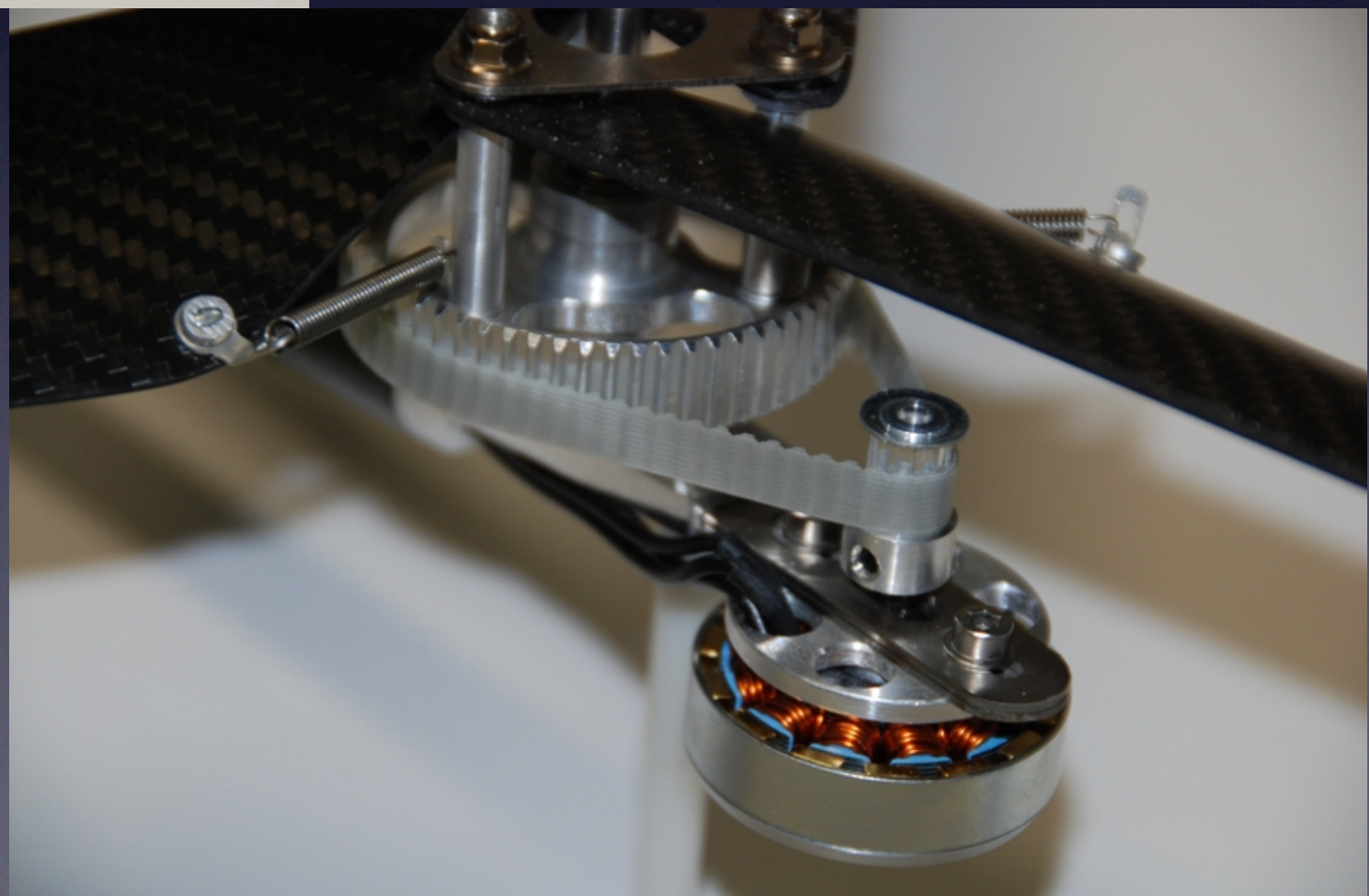
Quad-Rotor Helicopter



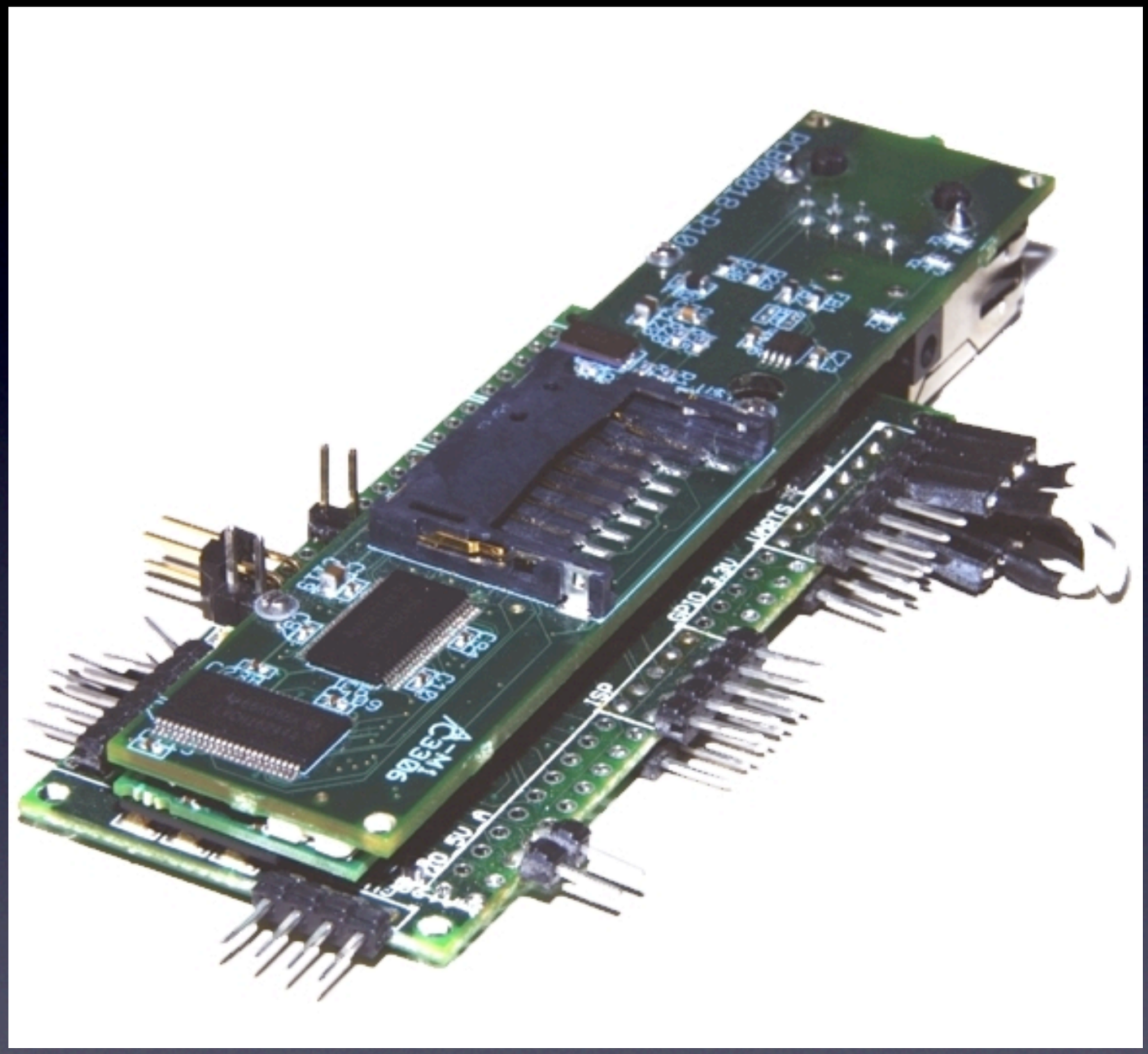


Gyro

Propulsion



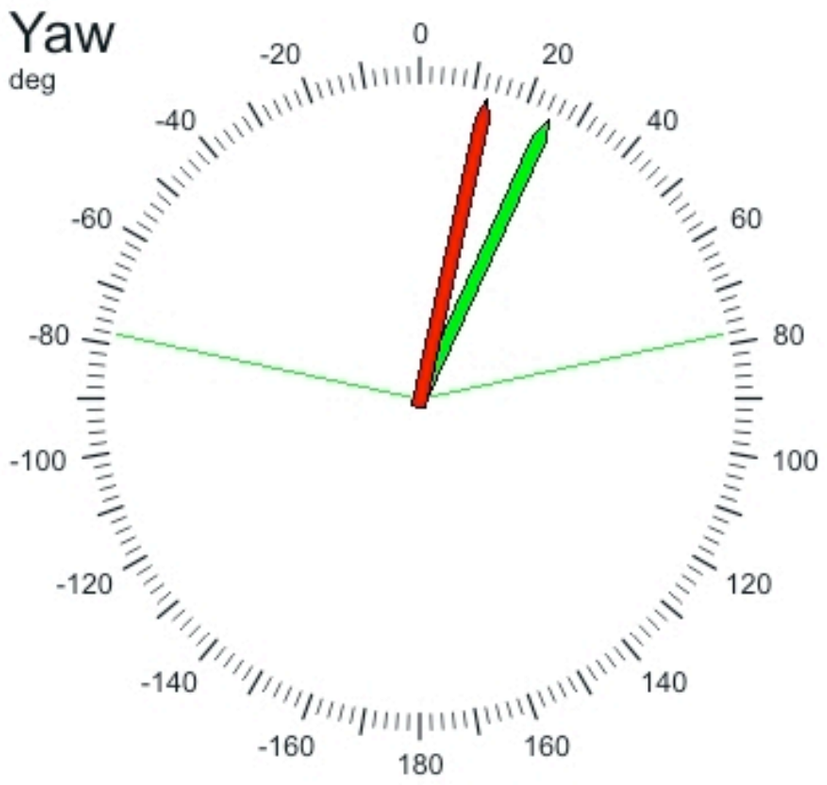
Gumstix



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

JAviator Control Terminal

− + Yaw Limit | Joystick Mode | Direct Connection | Connect to Heli | Start Test Mode | Start Logging | Roll Limit − +



Front

Left Hover Right

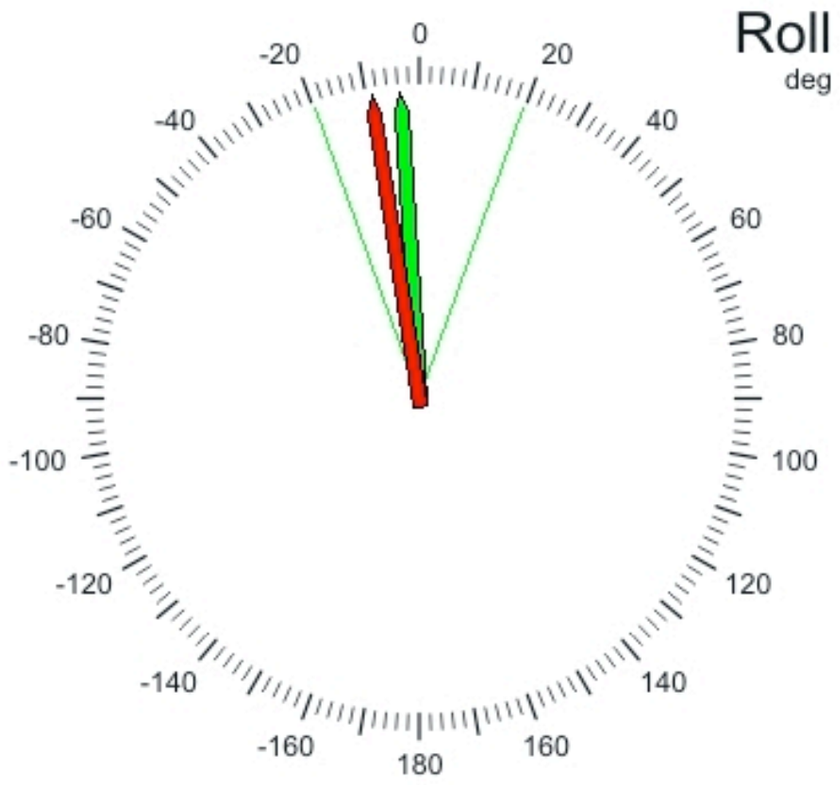
Rear

-1 -1 0

1 Offsets 1

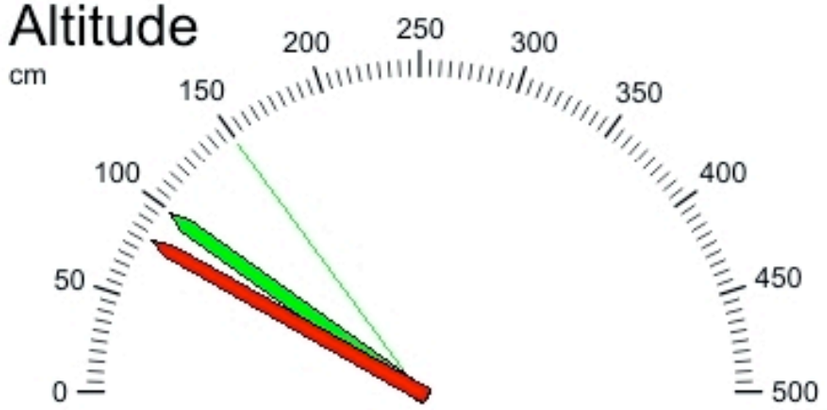
0 -1 0

353 Signals 352



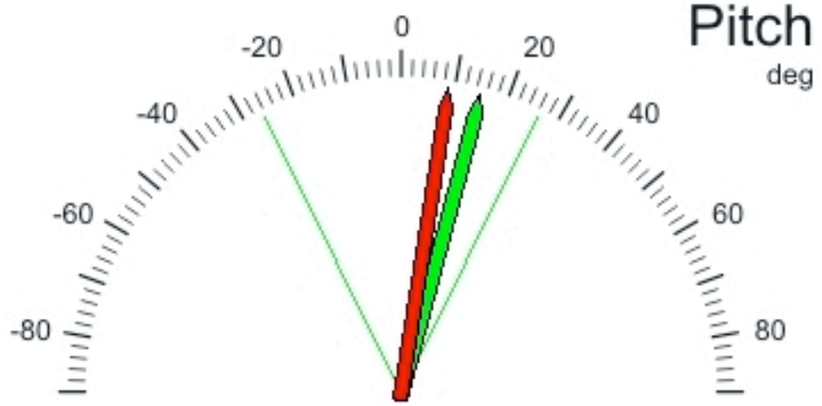
− + 33 Yaw Trim

Roll Trim 0 − +



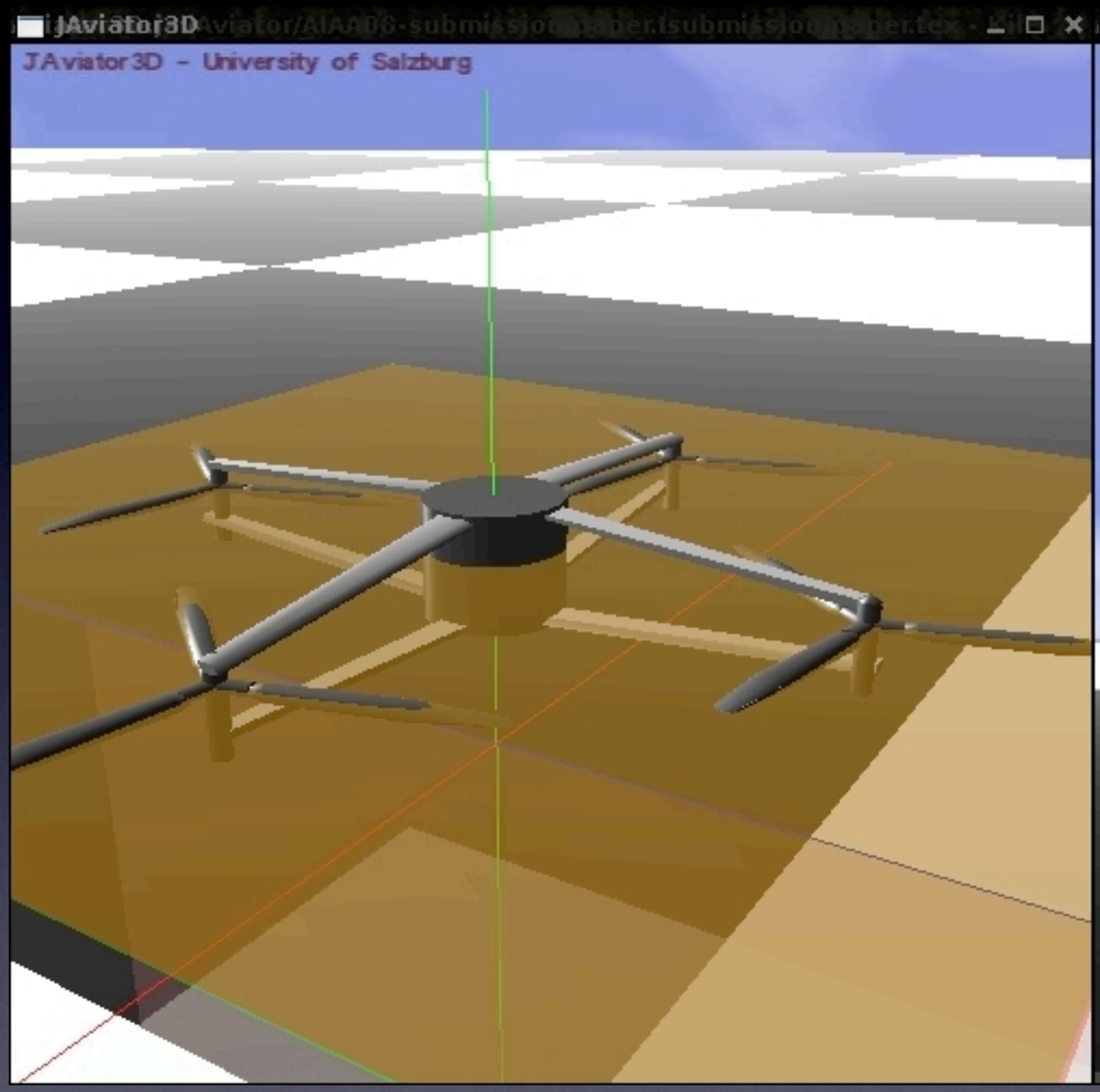
351

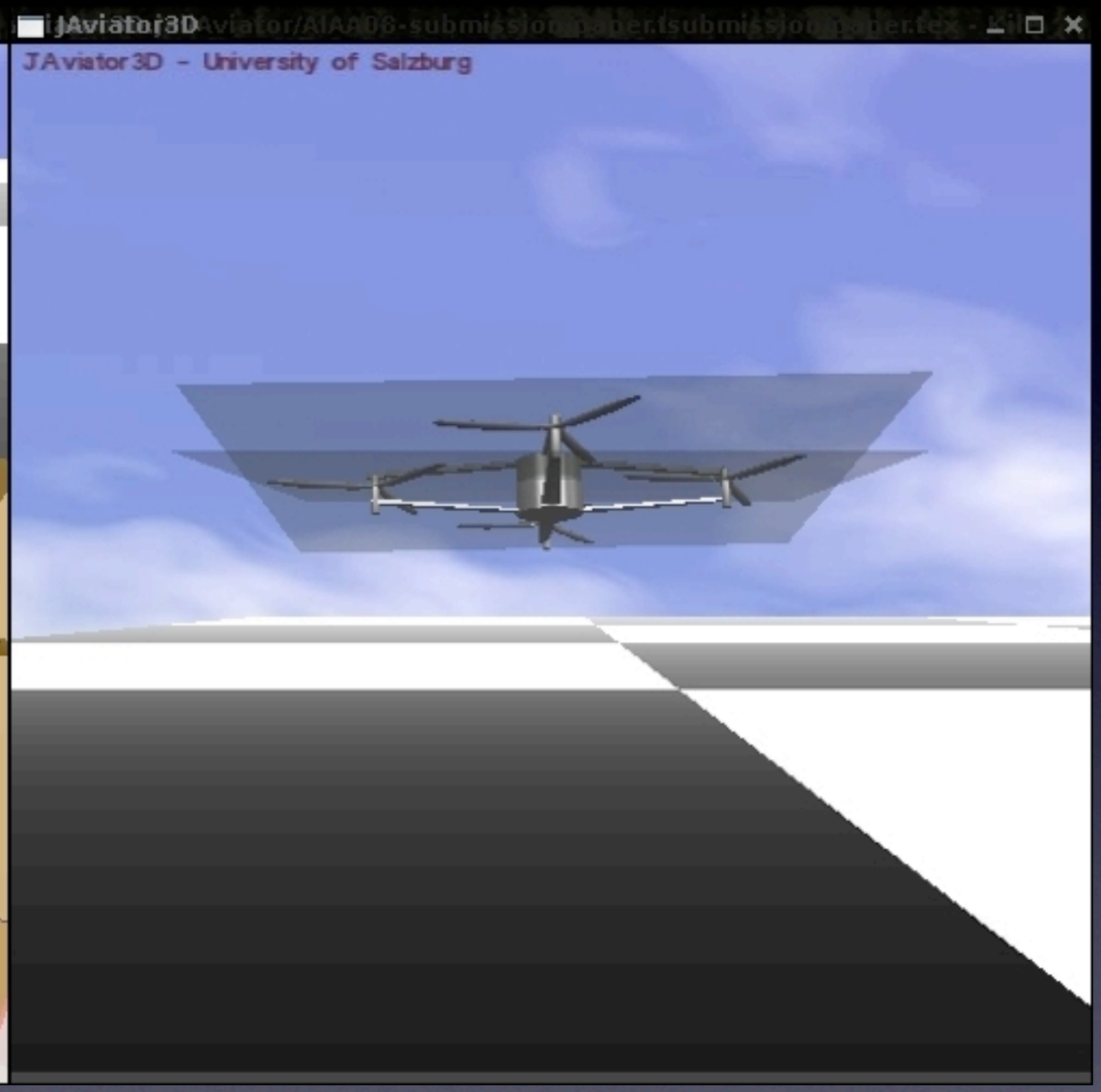
351



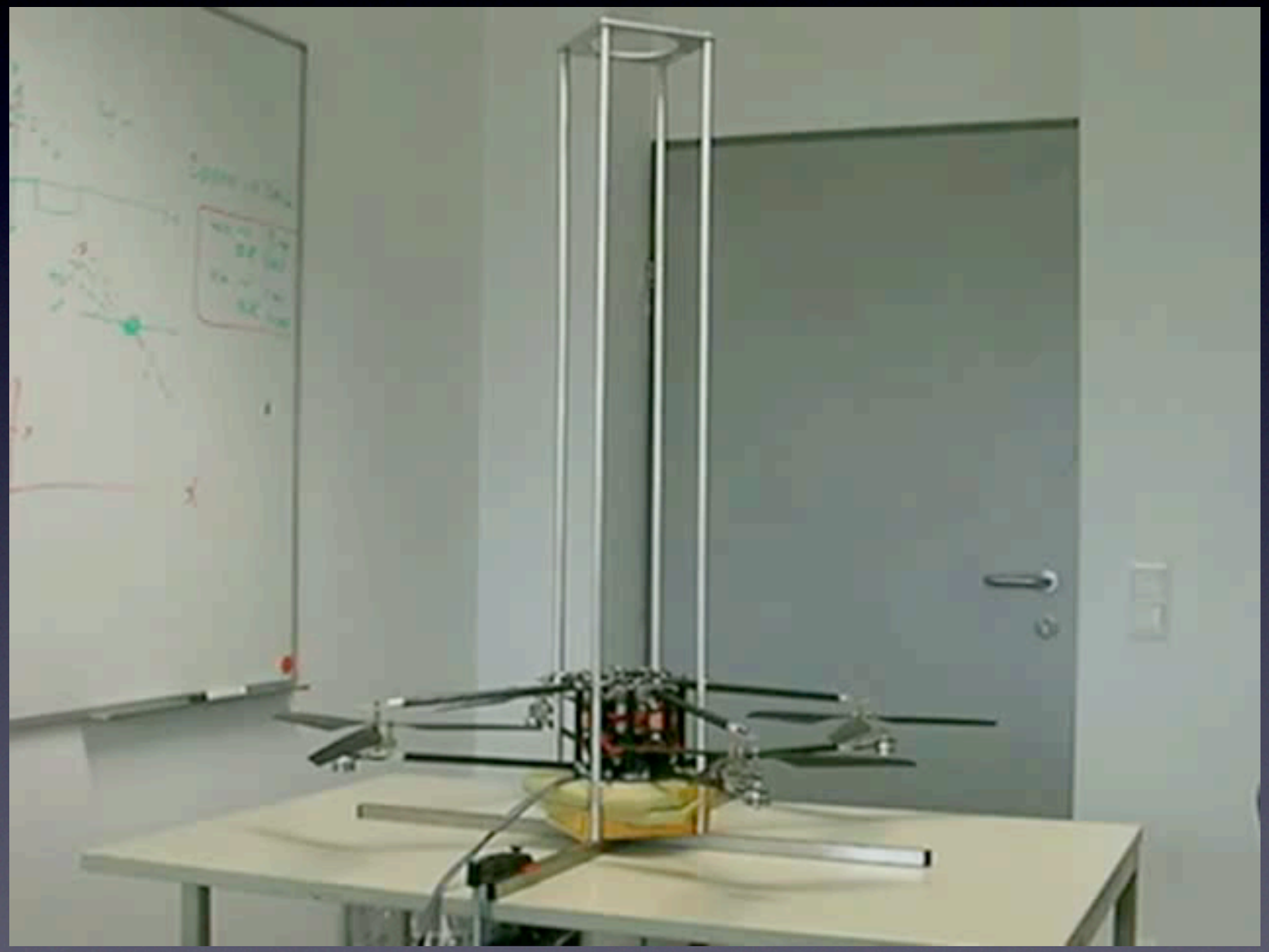
Pitch Trim 0 − +

− + Altitude Limit | Port Settings | Reset Needles | Shut down Heli | Key Assistance | About Terminal | Pitch Limit − +

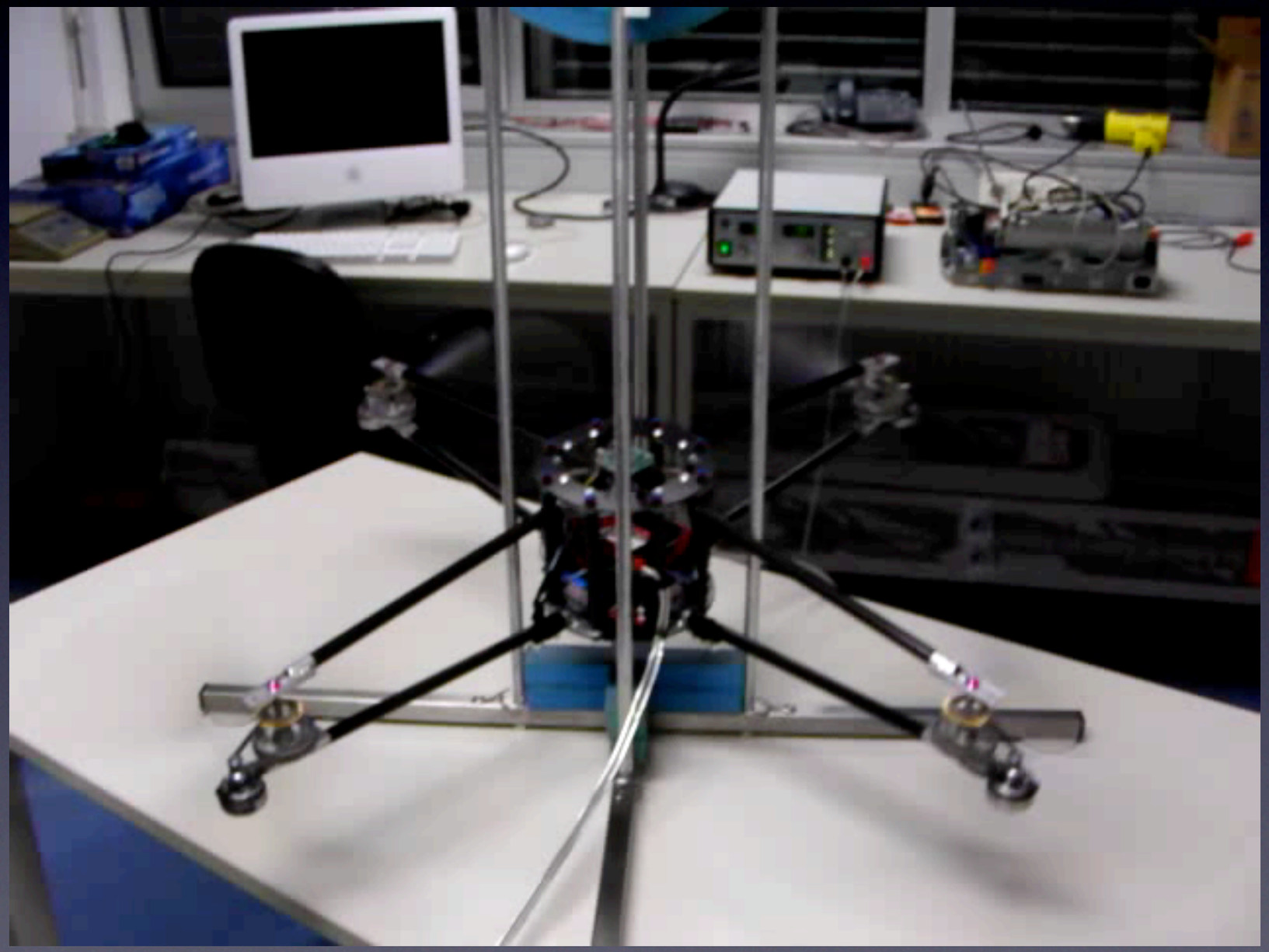




Oops



Flight Control

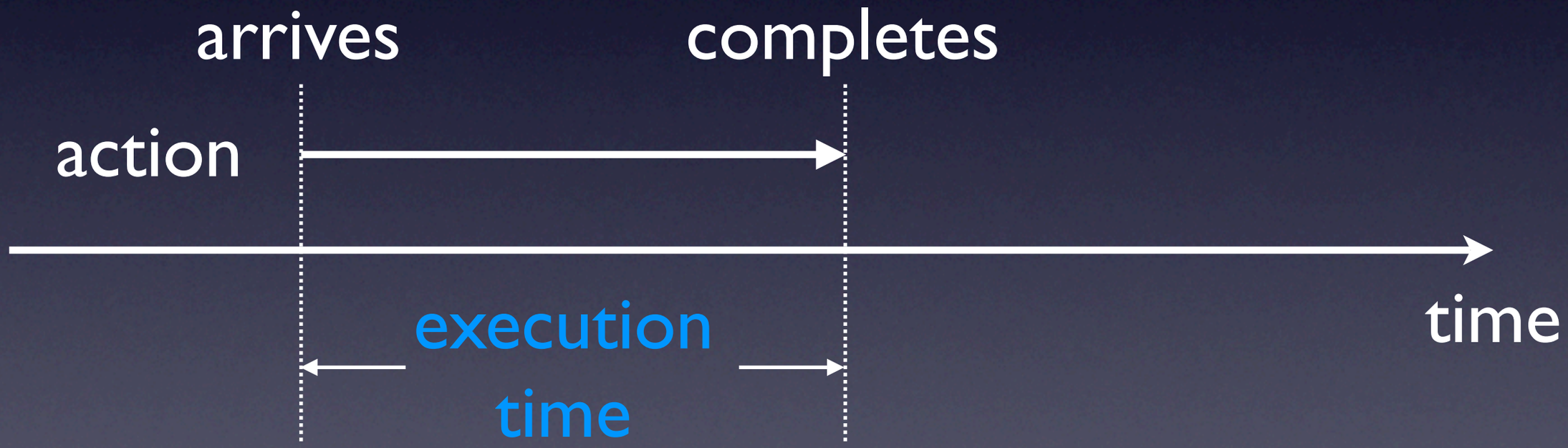


[AIAA GNC 2008]

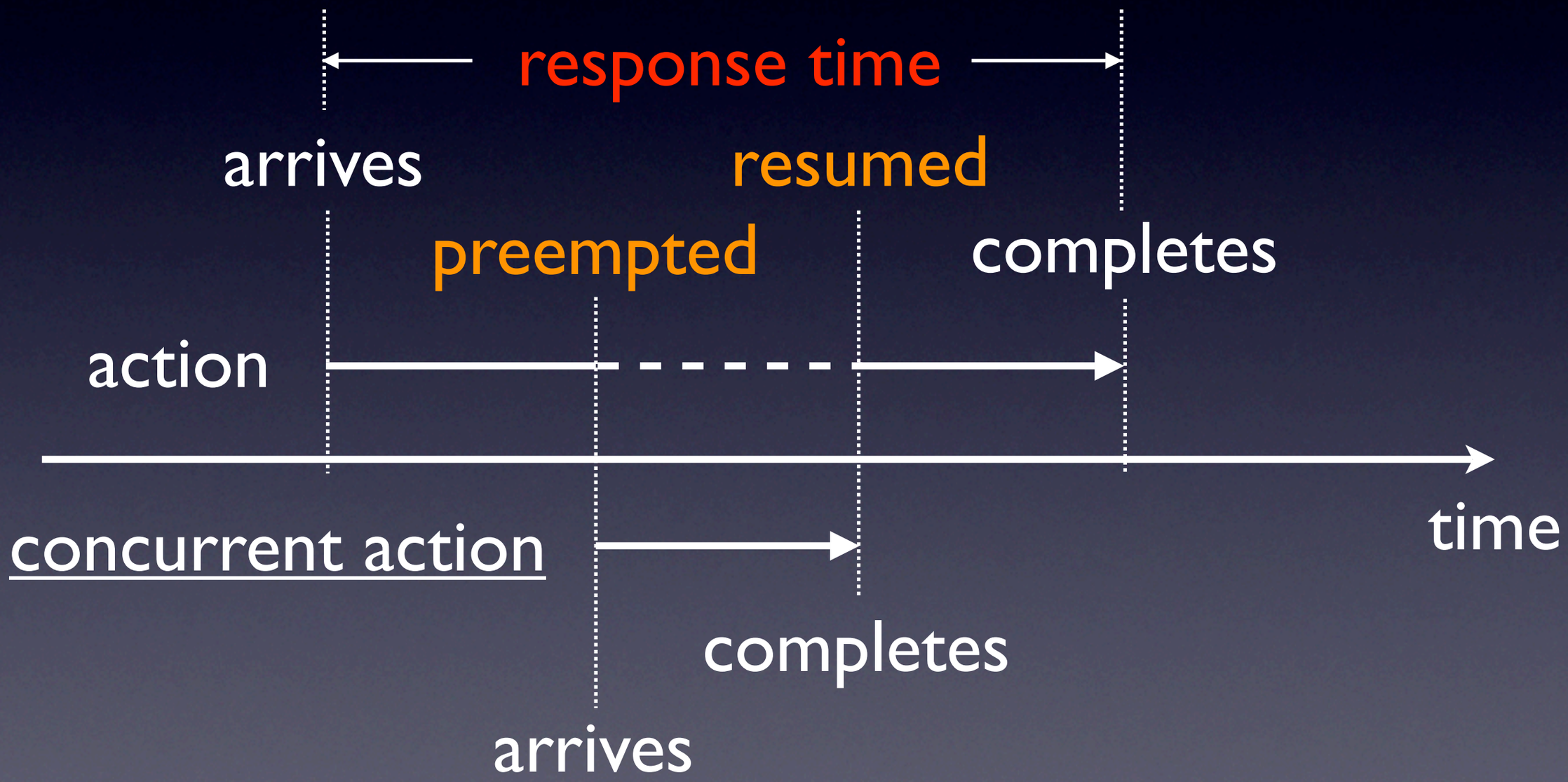
Outline

1. Time-Portable Programming
2. Tiptoe OS Scheduler
3. Tiptoe OS Memory Management

Process Action



Concurrency



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 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"

Time-Portable Programming

Giotto
[EMSOFT 2001, Proceedings of the IEEE 2003]

HTL
[EMSOFT 2006]

Exotasks
[LCTES 2007, TECS 2008]

Tiptoe
[USENIX 2008]

Outline

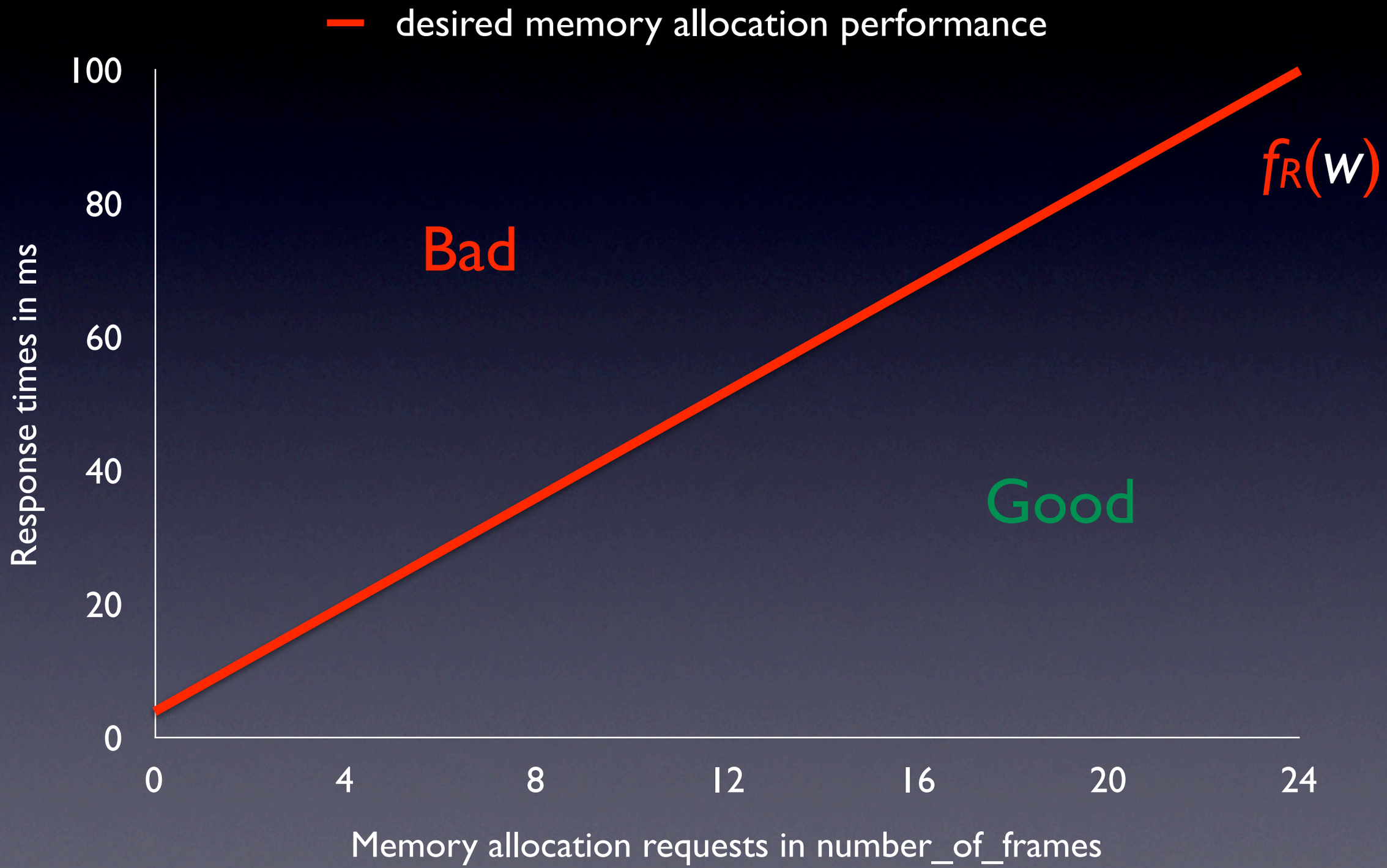
1. Time-Portable Programming
2. Tiptoe OS Scheduler
3. Tiptoe OS Memory Management

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

Response-Time Function



Throughput & Latency

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

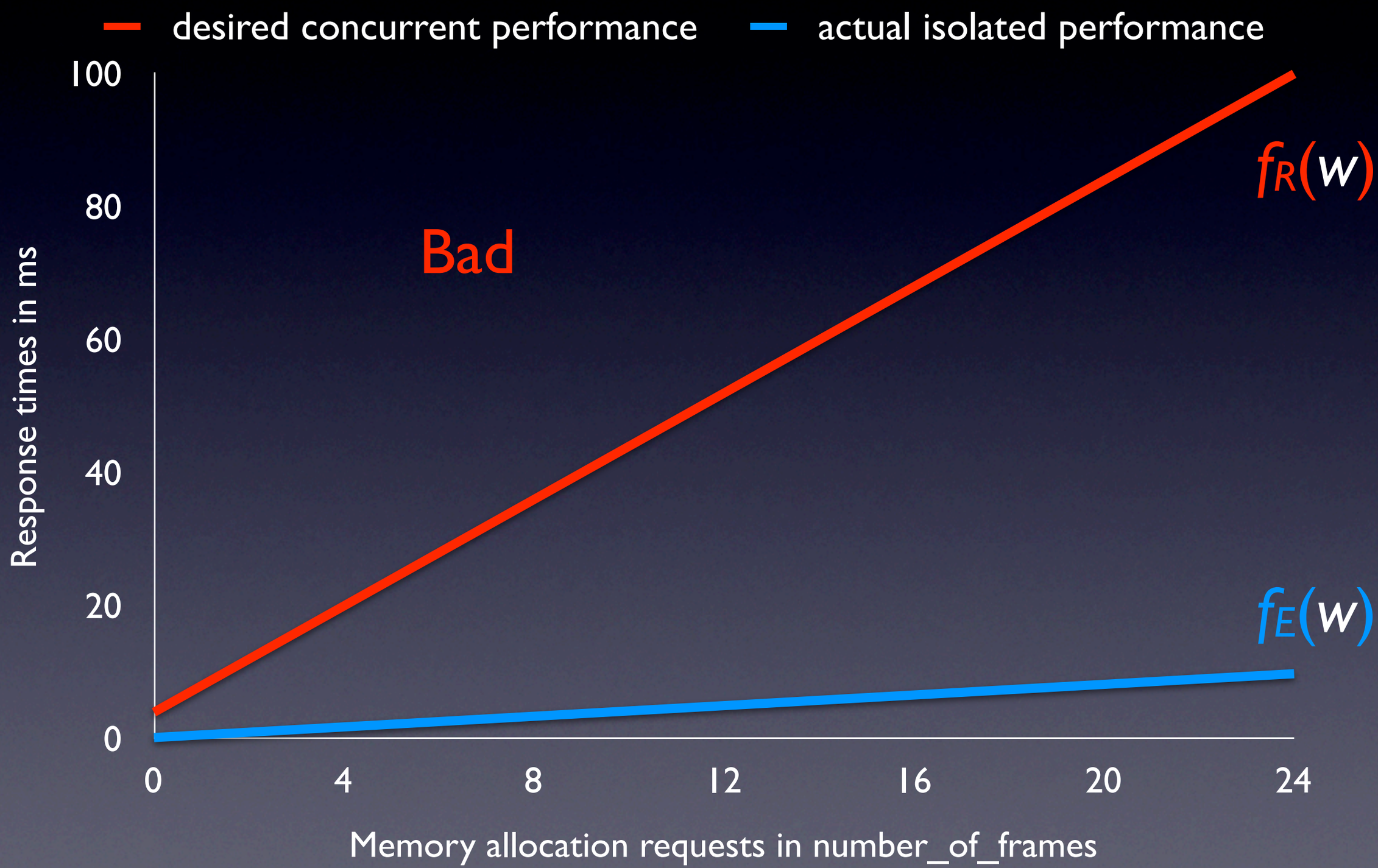
...

f_R(4 frames) = 20ms yields 200fps

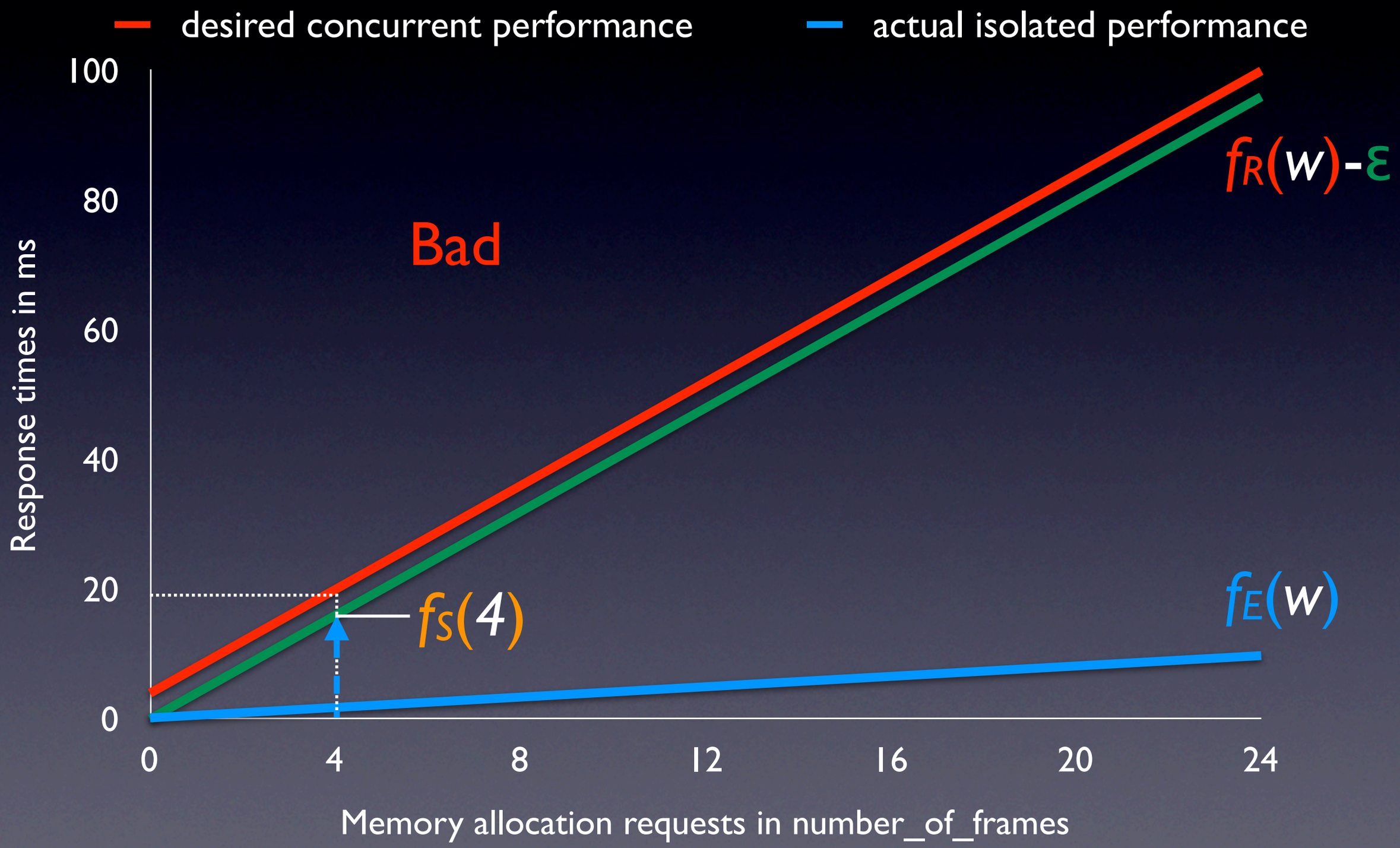
...

f_R(24 frames) = 100ms yet 240fps

Execution-Time Function



Scheduled Response Time



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

and

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

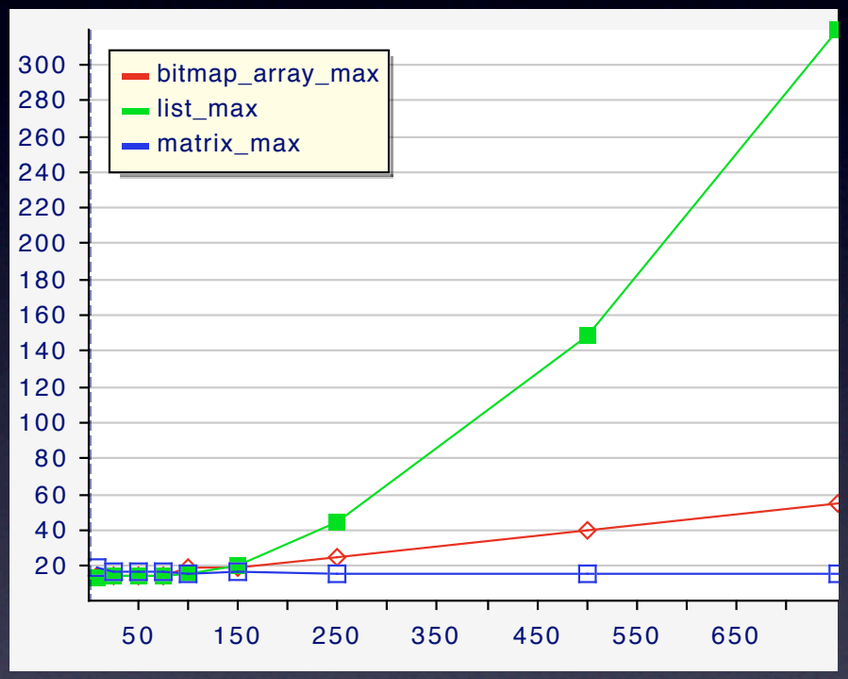
with ε representing the

"degree of time portability"

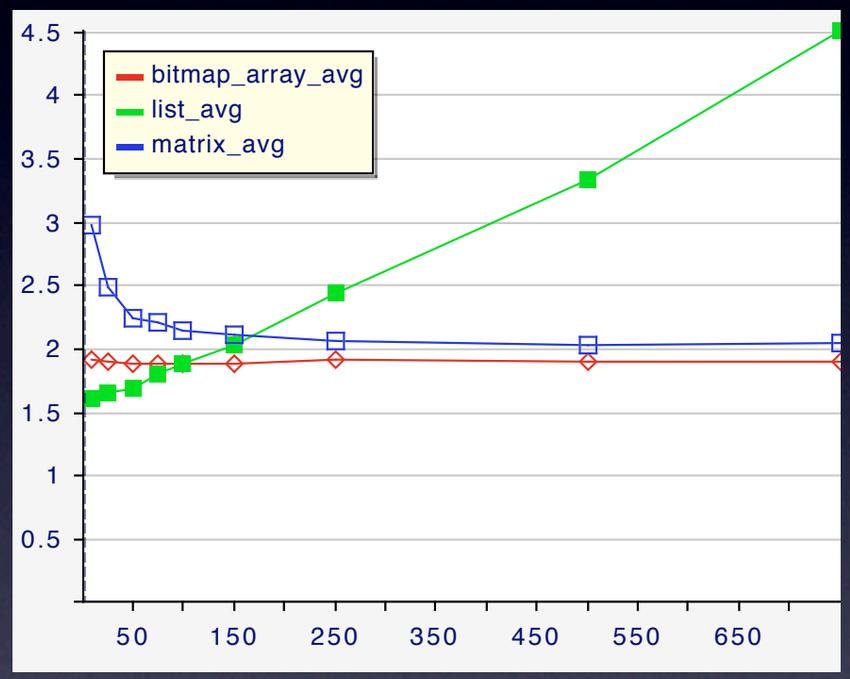
Scheduling Algorithm

- 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

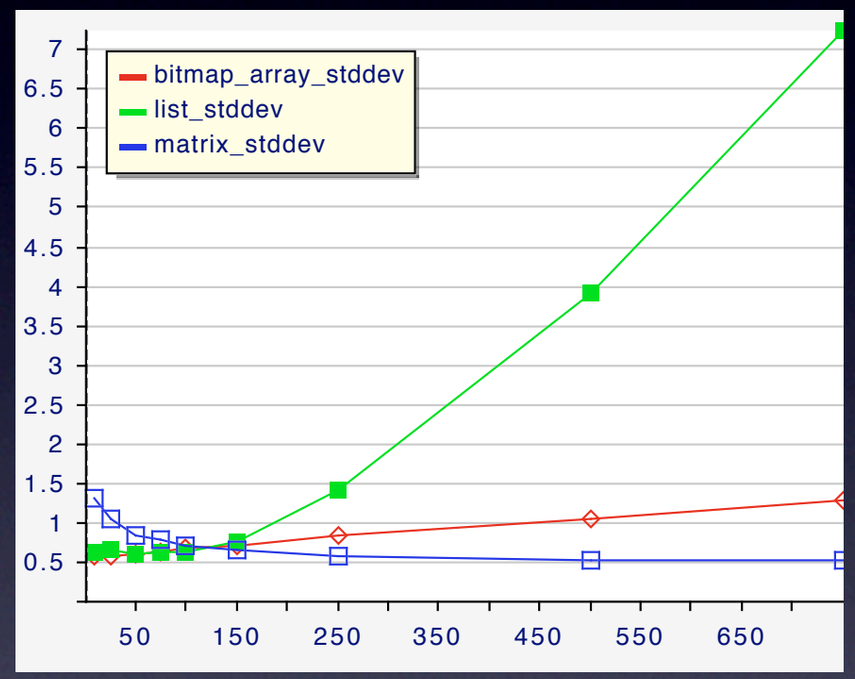
Scheduler Overhead



Max

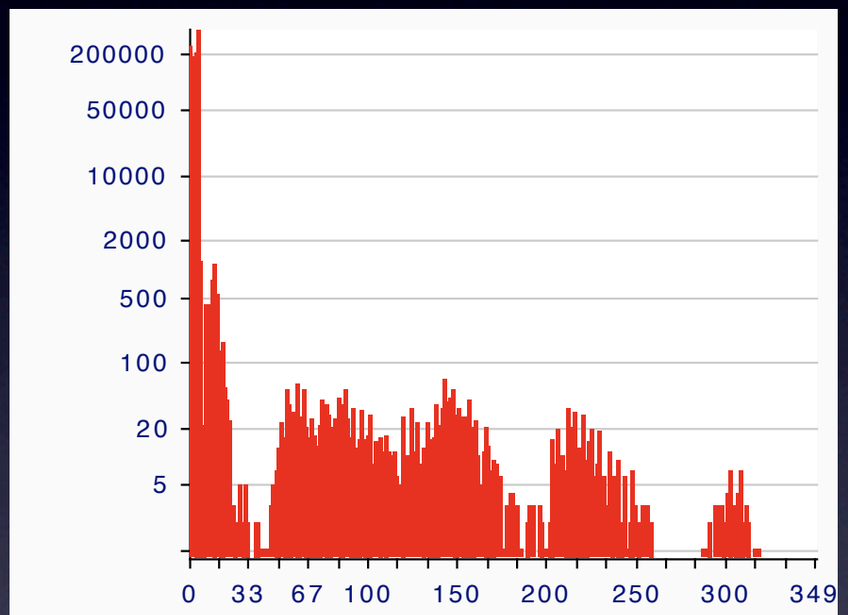


Average

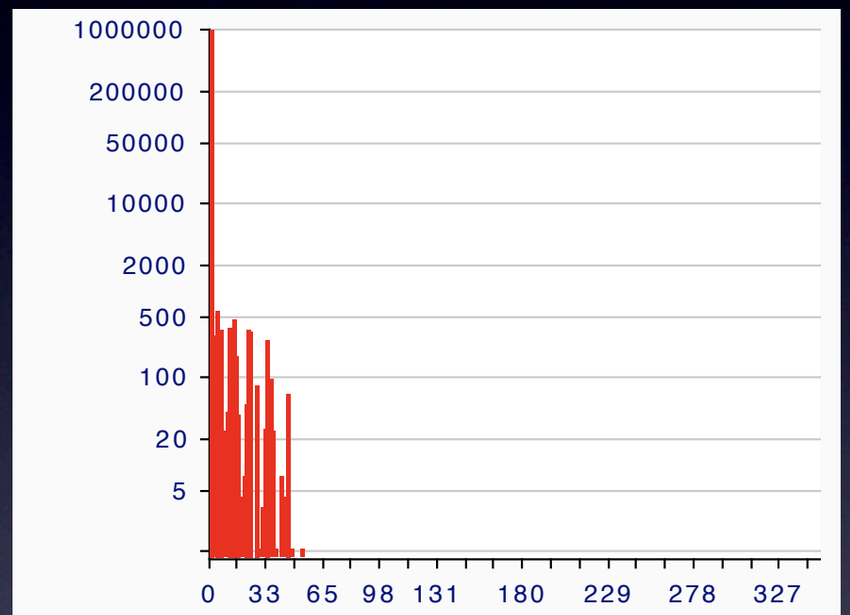


Jitter

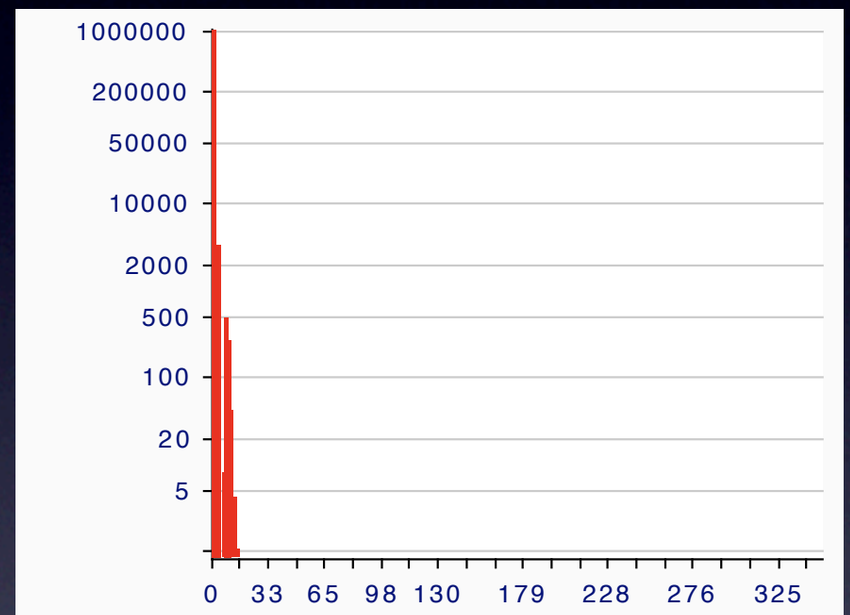
Execution Time Histograms



List

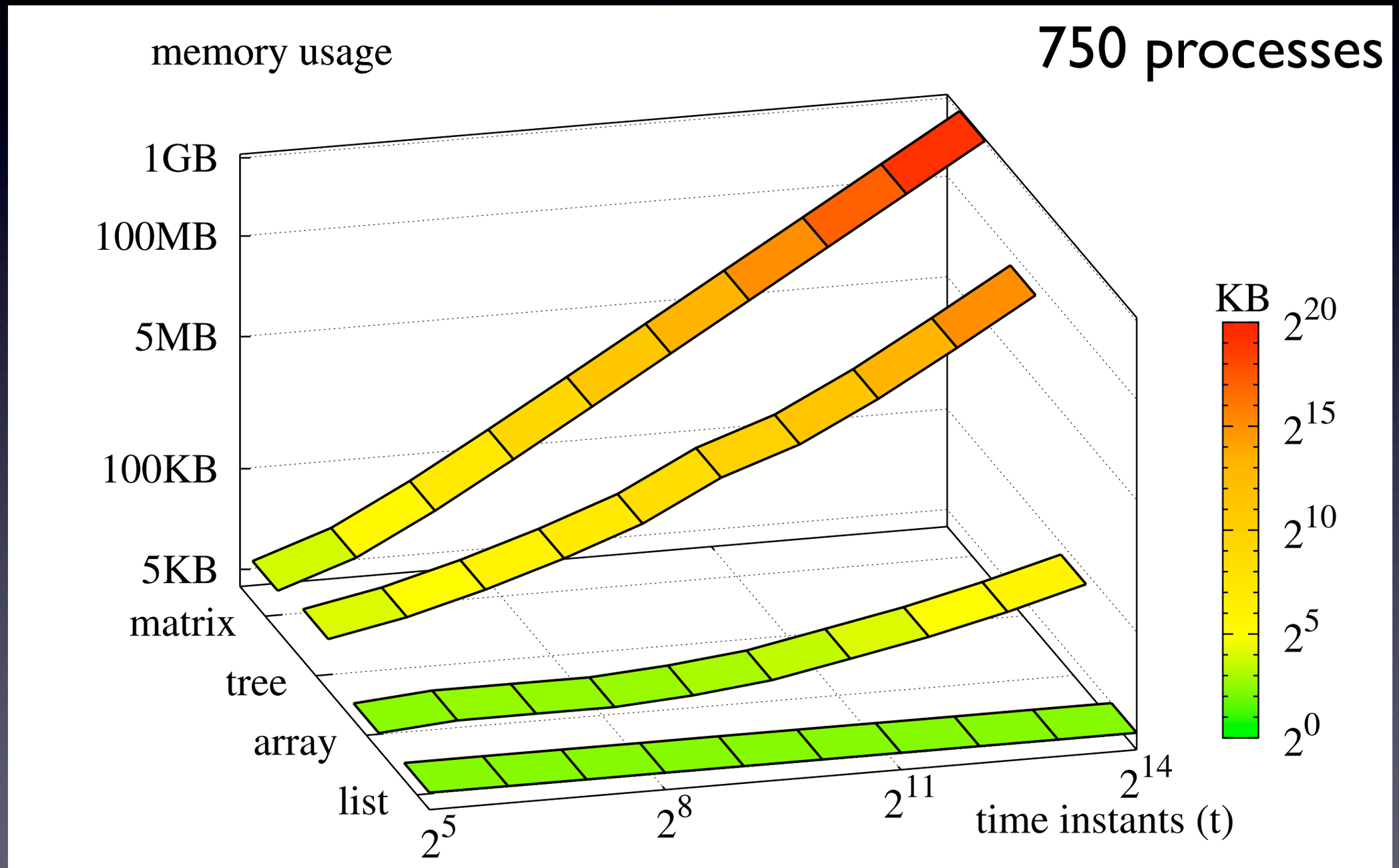


Array



Matrix

Memory Overhead



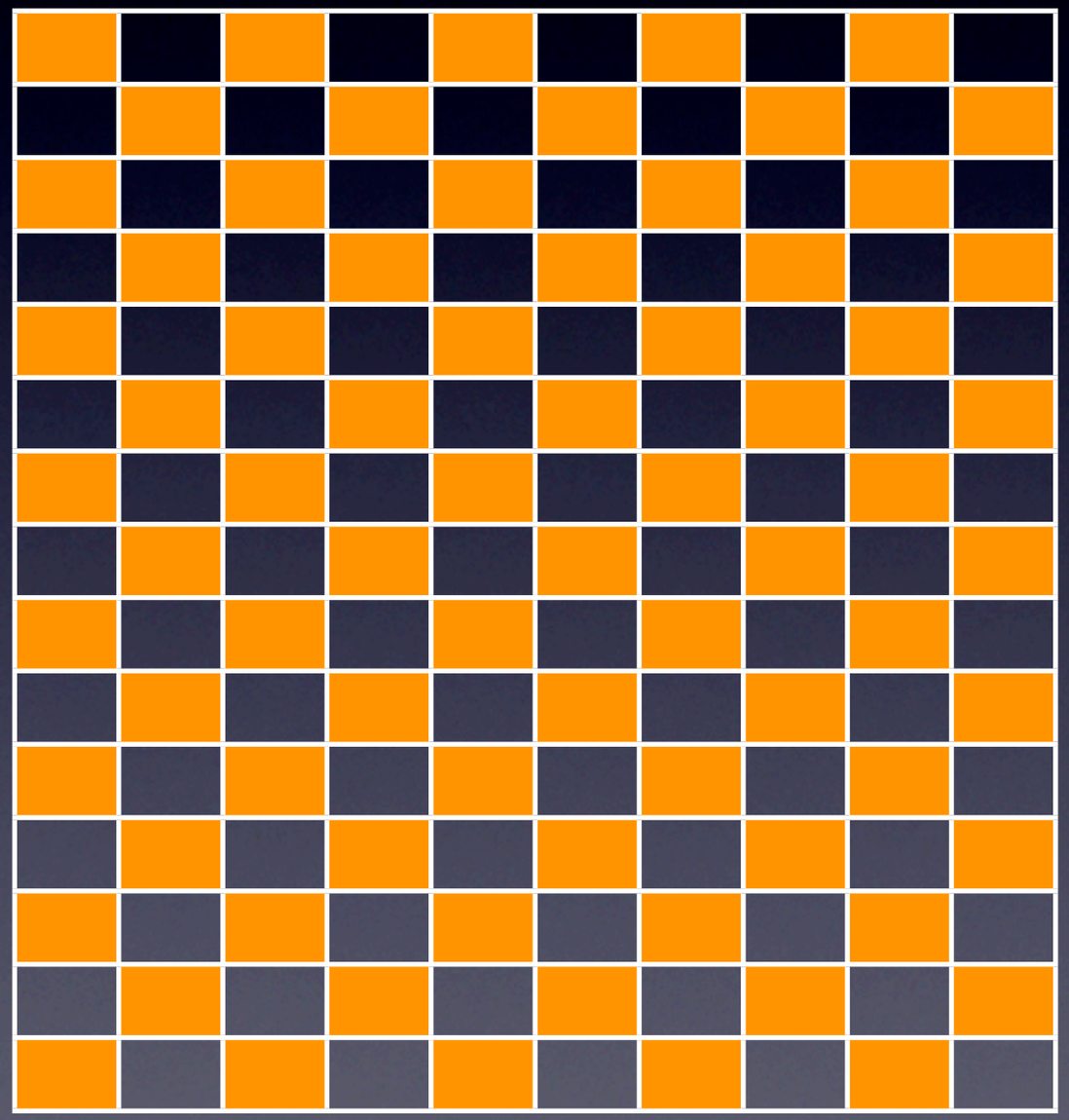
Outline

1. Time-Portable Programming
2. Tiptoe OS Scheduler
3. Tiptoe OS Memory Management

“Compact-Fit” [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

The Problem

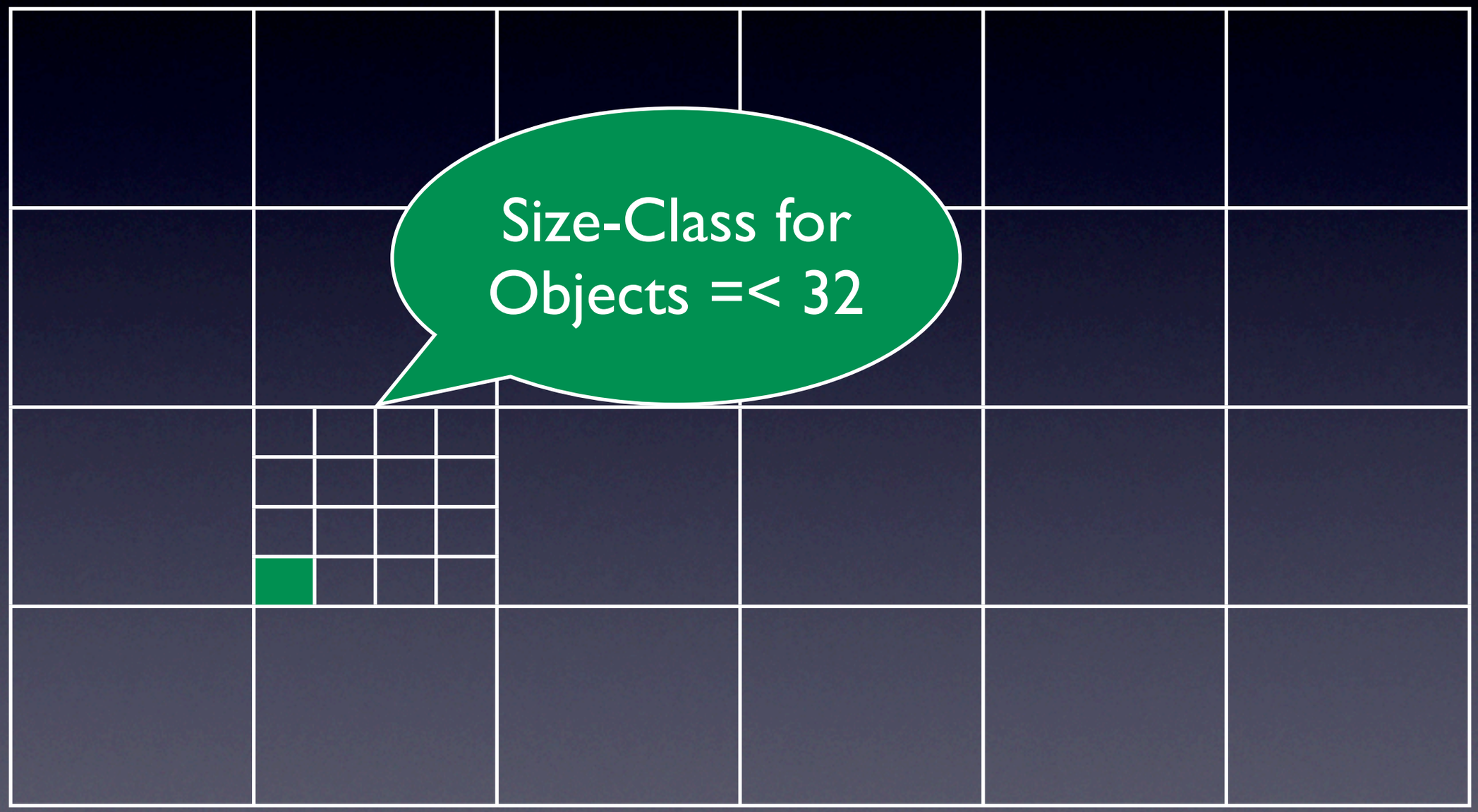


- Fragmentation
  - Compaction
- References
  - Abstract Space

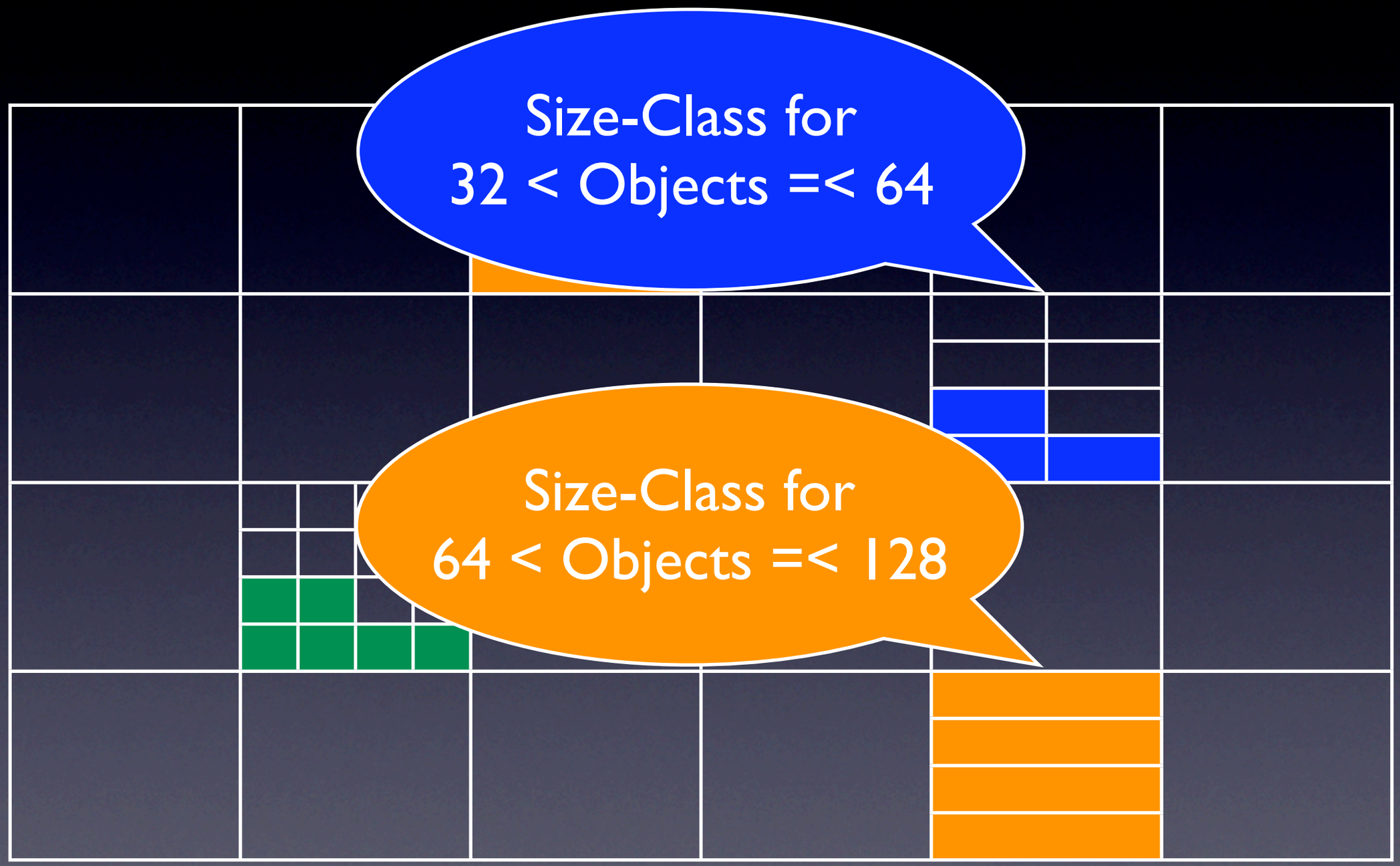
Partition Memory into Pages

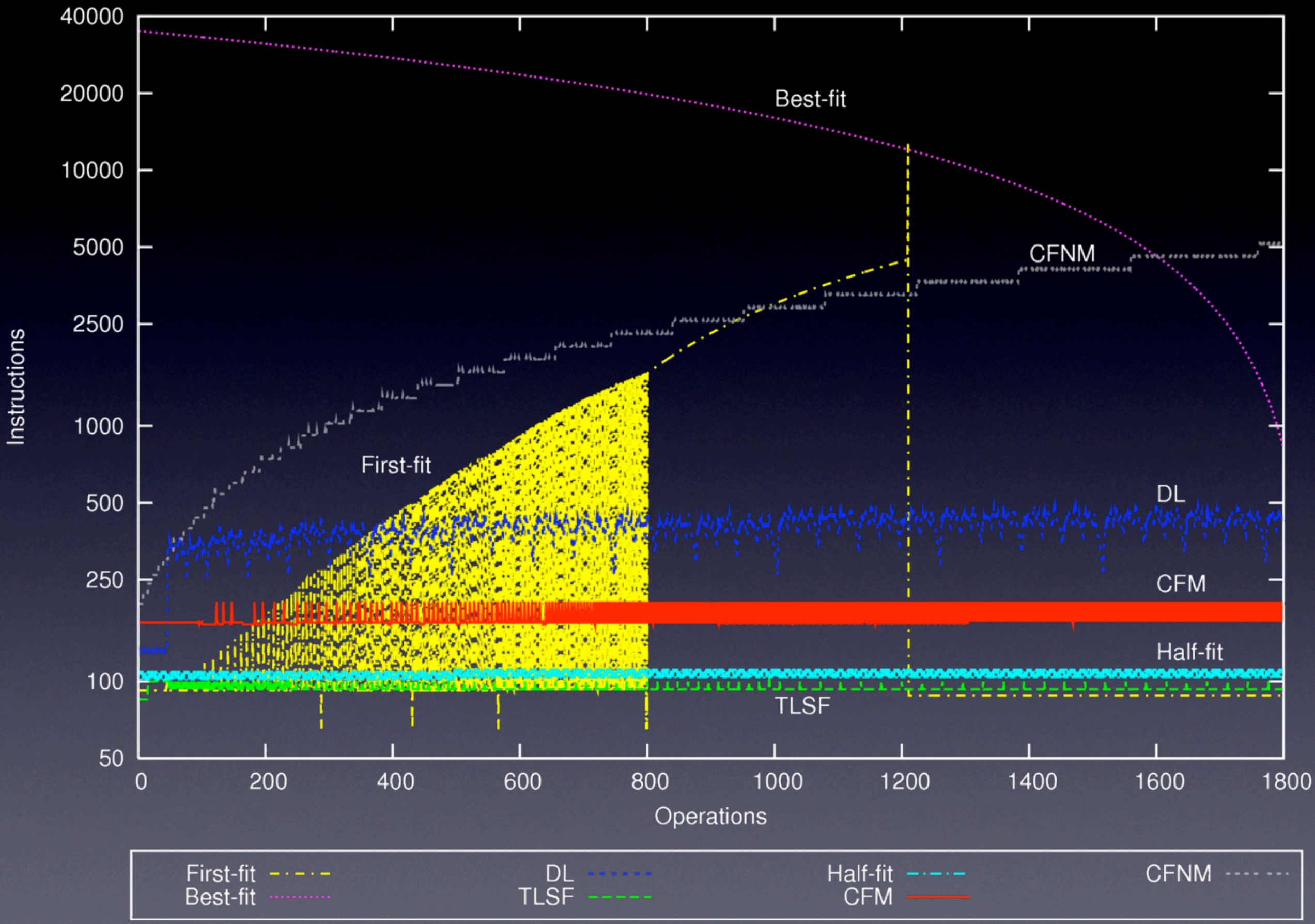
16KB	16KB	16KB	16KB	16KB	16KB
16KB	16KB	16KB	16KB	16KB	16KB
16KB	16KB	16KB	16KB	16KB	16KB
16KB	16KB	16KB	16KB	16KB	16KB

Partition Pages into Blocks



Size-Classes

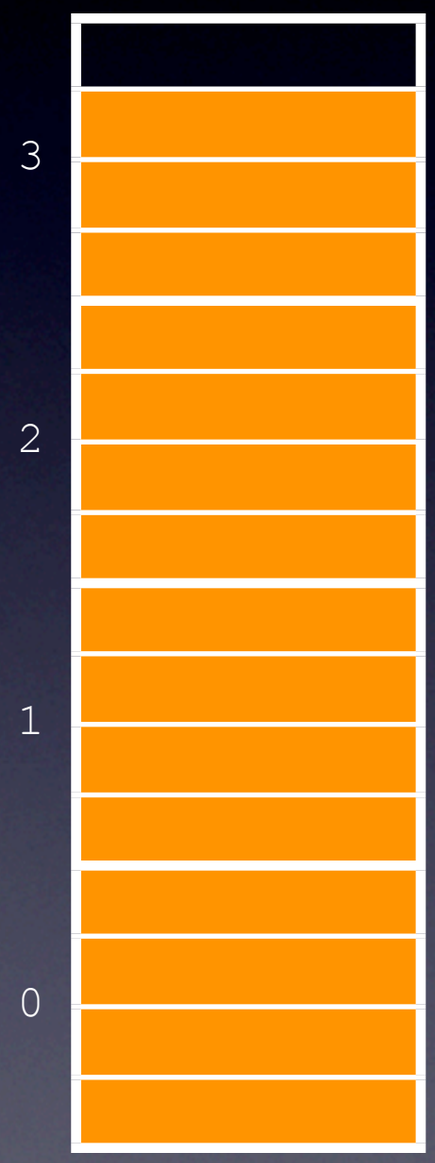



40000
20000
10000
5000
2500
1000
500
250
100
50
Instructions
0
200
400
600
800
1000
1200
1400
1600
1800
Operations
Best-fit
CFNM
First-fit
DL
CFM
Half-fit
TLSF
First-fit
Best-fit
DL
TLSF
Half-fit
CFM
CFNM

Invariant: Size-Class Compact

Objects =< 32

Objects =< 64



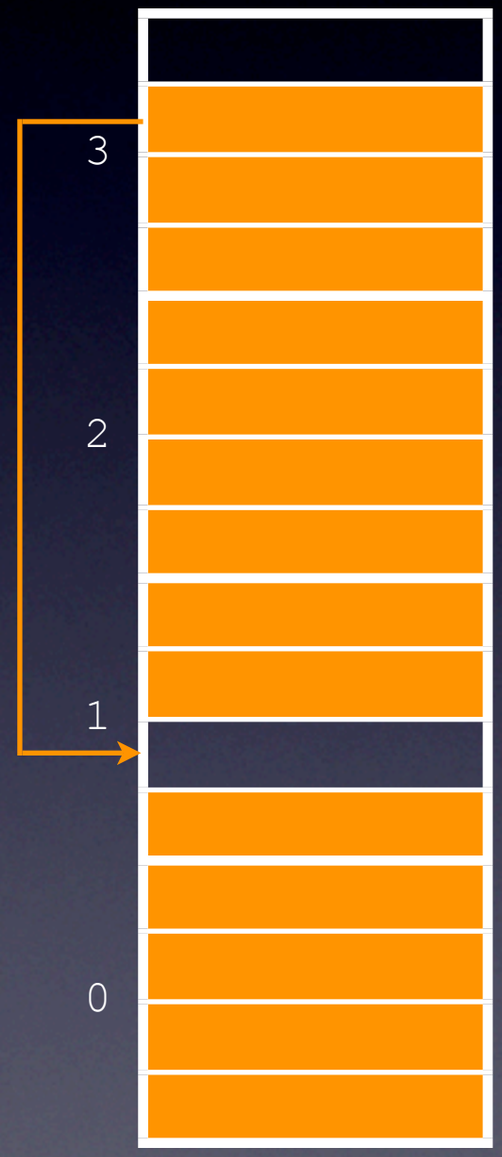
Objects =< 128

“Compact-Fit” (Bounded Compaction)

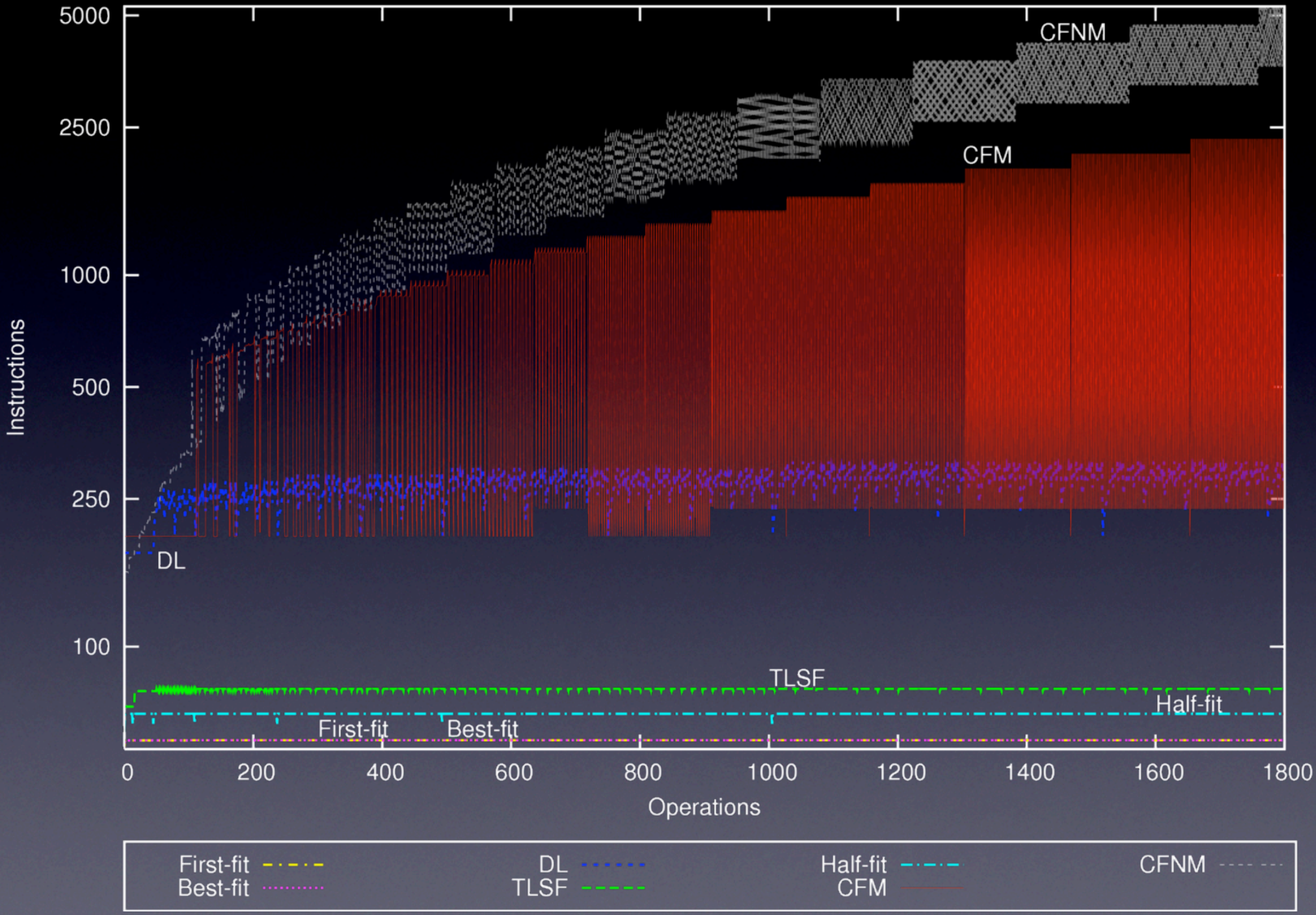
just move ‘last’ object

Objects =< 32

Objects =< 64



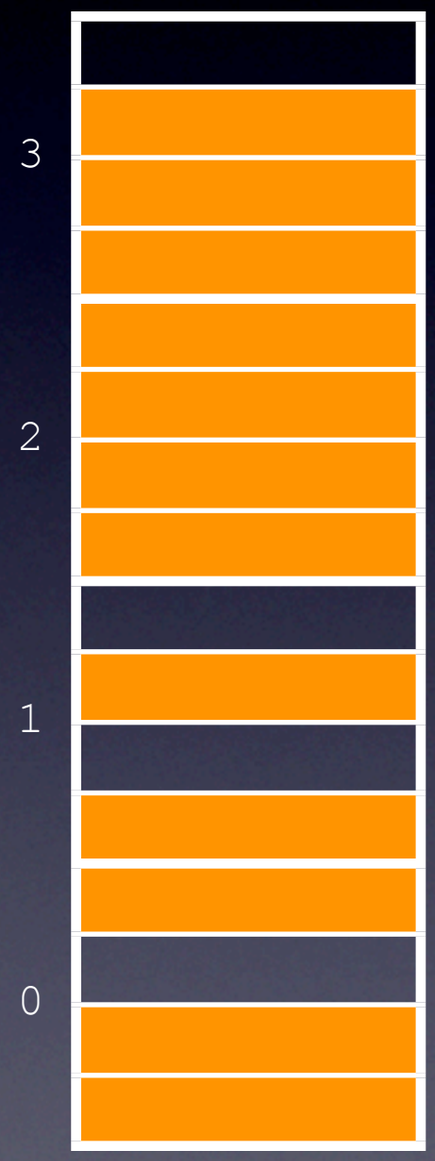
Objects =< 128


5000
2500
1000
500
250
100
Instructions
CFNM
CFM
DL
TLSF
Half-fit
First-fit
Best-fit
0
200
400
600
800
1000
1200
1400
1600
1800
Operations
First-fit
Best-fit
DL
TLSF
Half-fit
CFM
CFNM

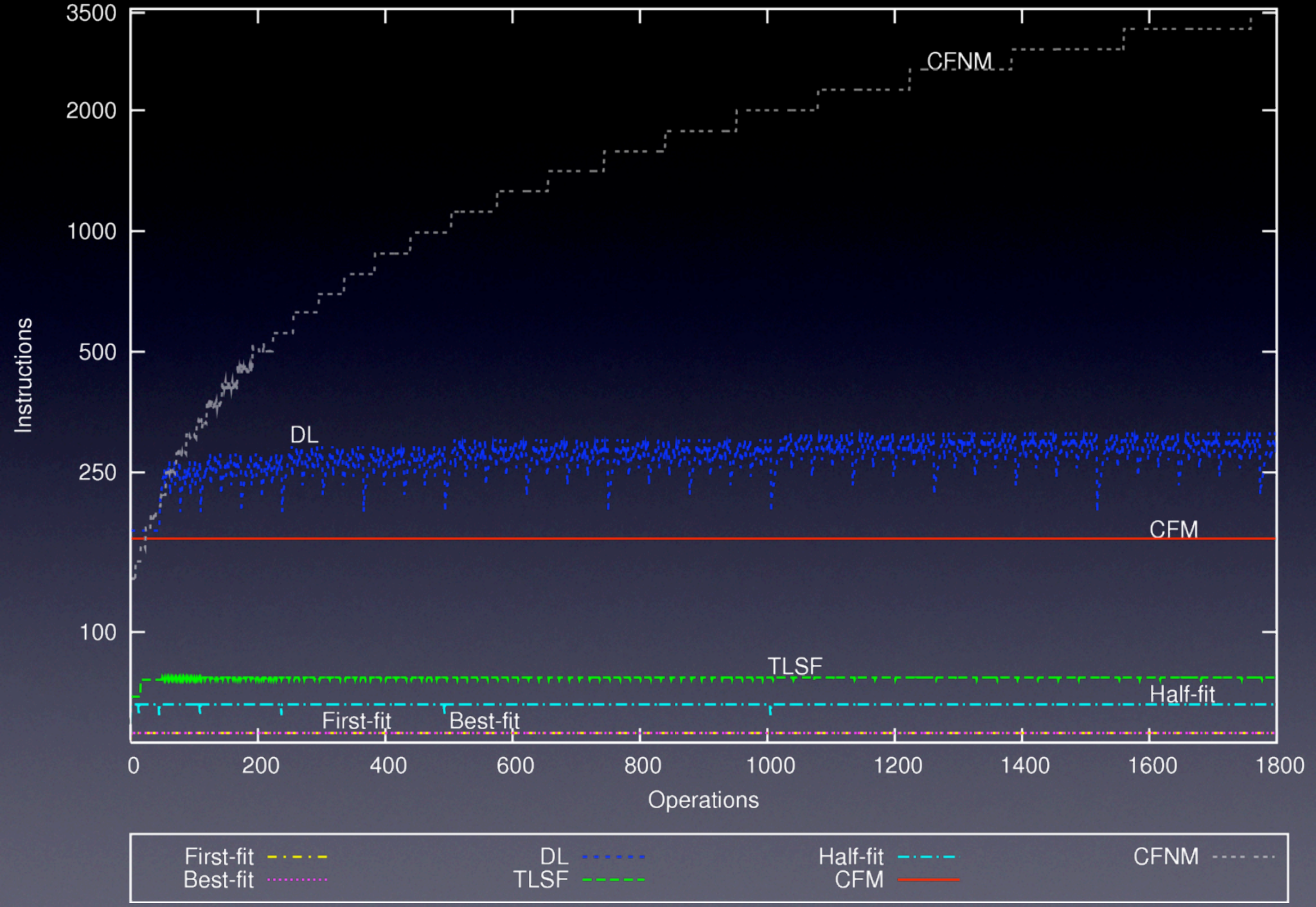
Partial Compaction

Objects =< 32

Objects =< 64



Objects =< 128


3500
2000
1000
500
250
100
Instructions
CFNM
DL
CFM
TLSF
Half-fit
First-fit
Best-fit
0
200
400
600
800
1000
1200
1400
1600
1800
Operations
First-fit
Best-fit
DL
TLSF
Half-fit
CFM
CFNM

Current/Future Work

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

Thank you