# Elastic Grid Reservations with User-Defined Optimization Policies

Workshop on Adaptive Grid Middleware, 30th September 2004

**Thomas Röblitz, Florian Schintke and Jan Wendler**

{roeblitz,schintke,wendler}@zib.de

**Zuse Institute Berlin**

# Why do we need reservations?

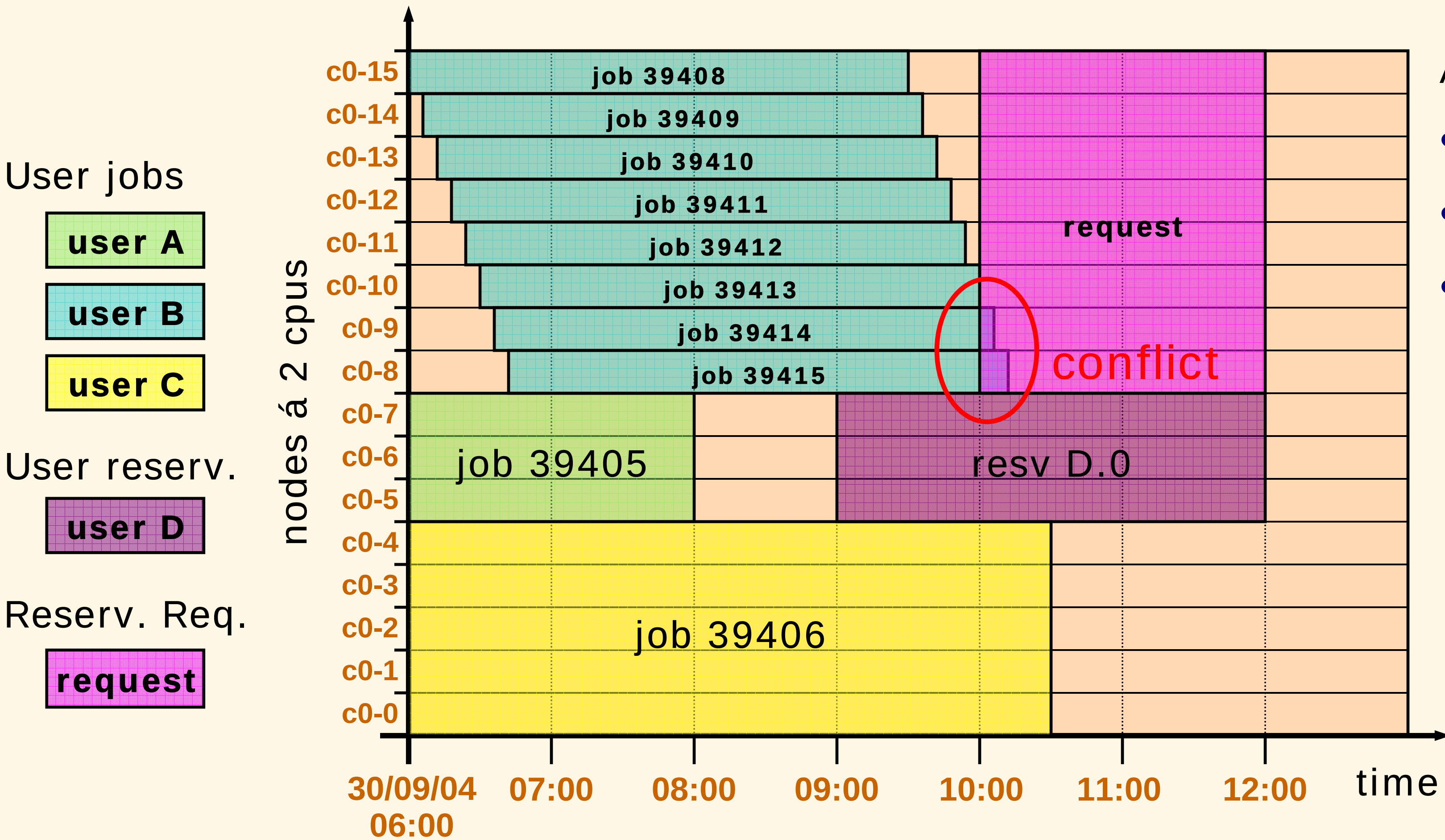
*Higher guarantee that specific resource allocations succeed!*

- Hotel reservation, ticket reservation, ...
- CPU reservation, bandwidth reservation, ...
- Reservations have been already studied in many areas incl. network management, CPU cycle scheduling and Grid computing.

➔ What's still to be done? *Flexibility & Efficiency*

# How do we reserve? – Rigid Parameters

Rigid request: Reserve 16 CPUs at cluster ELFIE from 30/09/04 10:00 til 30/09/04 12:00.

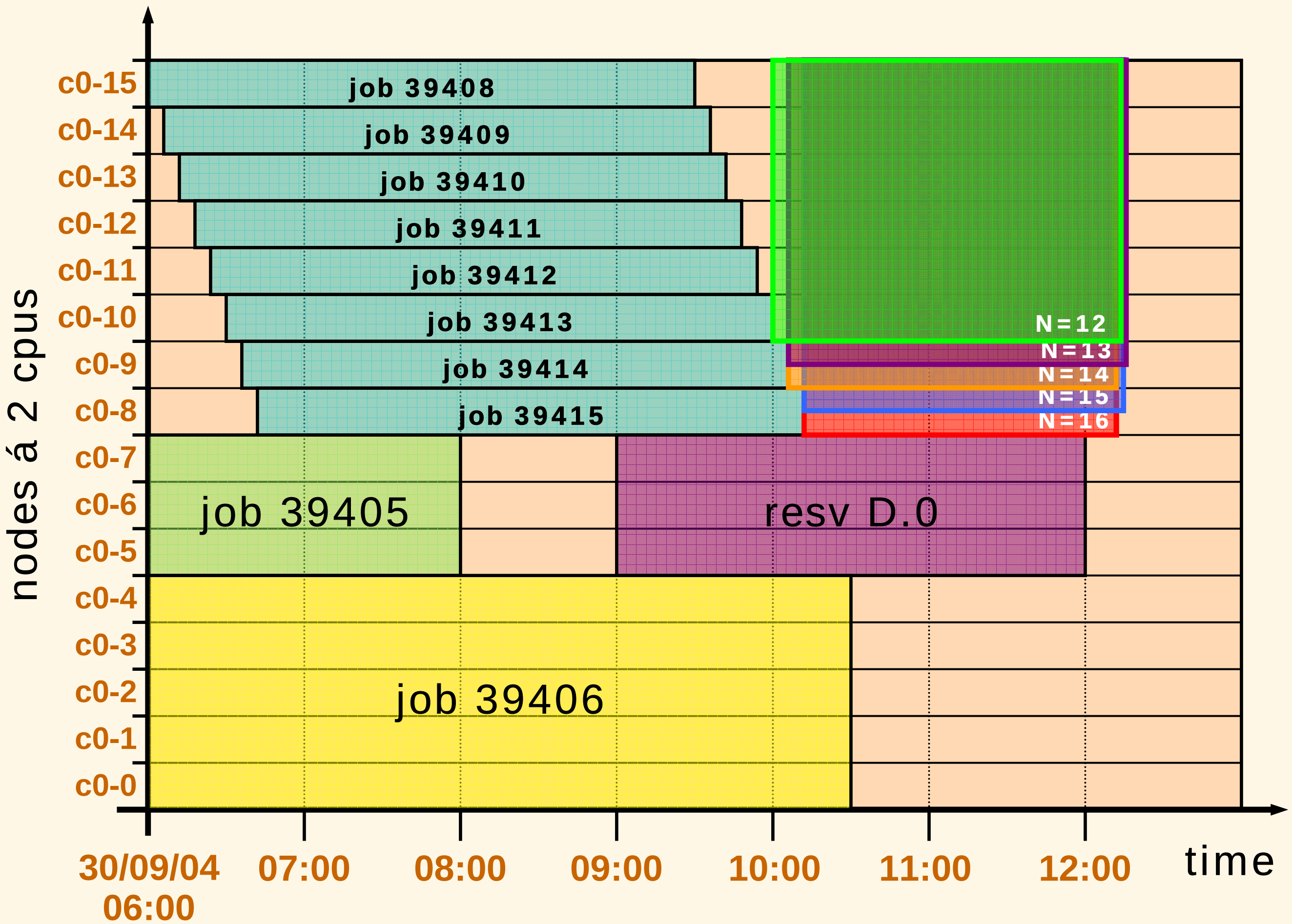



Alternatives:

- location (cluster)
- start time
- number of CPUs, duration

# How do we reserve? – Time & CPU range

Elastic request: Reserve 12-16 CPUs at cluster ELFIE for 2 hours from 30/09/04 10:00 til 30/09/04 13:00 with s=0.1, p=0.9 (speedup parameters*).




Determine alternative duration:

- Assume 2 hours given for 16 CPUs ($dur_{ref}=2$, $N_{ref}=16$).
- $dur(N)=sp(N_{ref})*dur_{ref}/sp(N)$
- **dur(15) = 02:03h**, **dur(14) = 02:06h**,
- **dur(13) = 02:10h**, **dur(12) = 02:14h**

*Speedup *sp* defined as $sp(N)=1/(s+p/N)$; s - sequential part, p – parallel part. (G. Amdahl 1967, *Validity of the single-processor approach to achieving large scale computing capabilities*)

# How do we reserve? – All together

Flexible aspects of a reservation request:

- CPU range
- time range
- location

> Elastic request: Reserve 8-16 CPUs at ANY cluster for 2 hours from 30/09/04 10:00 til 30/09/04 13:00 with s=0.1, p=0.9 (speedup parameters). The duration is given for 16 P4 (2GHz) CPUs.

Attributes of a request:

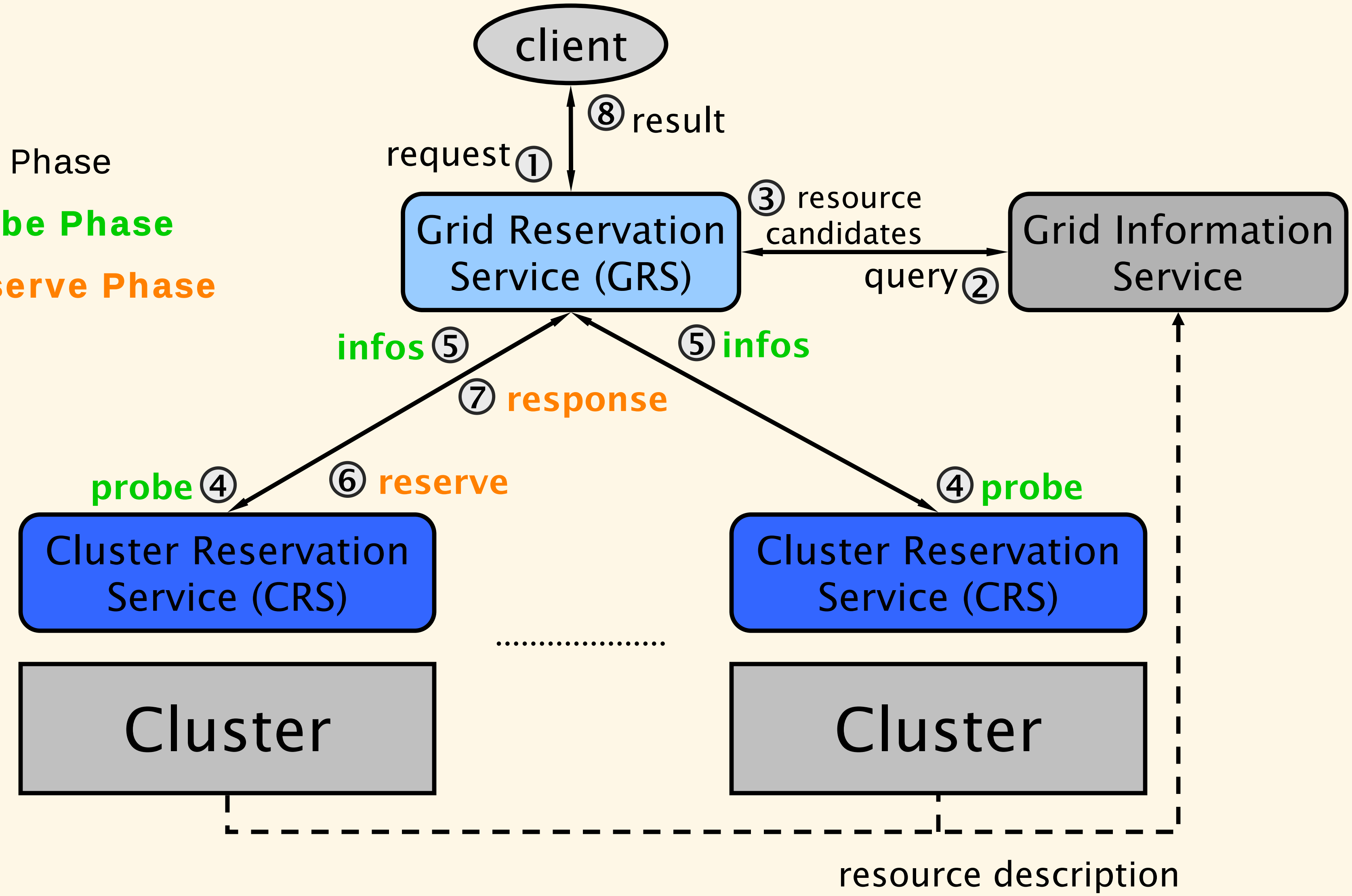
- CPU range: $np_{min}$, $np_{max}$
- time range: *earliest start time (est)*, *duration (dur)*, *latest end time (let)*
- CPU-time relation: *speedup (sp)*
- performance: $np_{ref}$, $dur_{ref}$, $perf_{ref}$
- miscellaneous, e.g. uid/gid/certificate

# Algorithm – Goals

- efficiency – number of reservation tries in case of success/failure
  - ➔ tradeoff between flexibility and reservation tries
- impact on user jobs – longer wait time
  - ➔ reservations reduce the optimization space for a local scheduler

# Algorithm – Services & Interactions

1. GIS Phase
2. **Probe Phase**
3. **Reserve Phase**

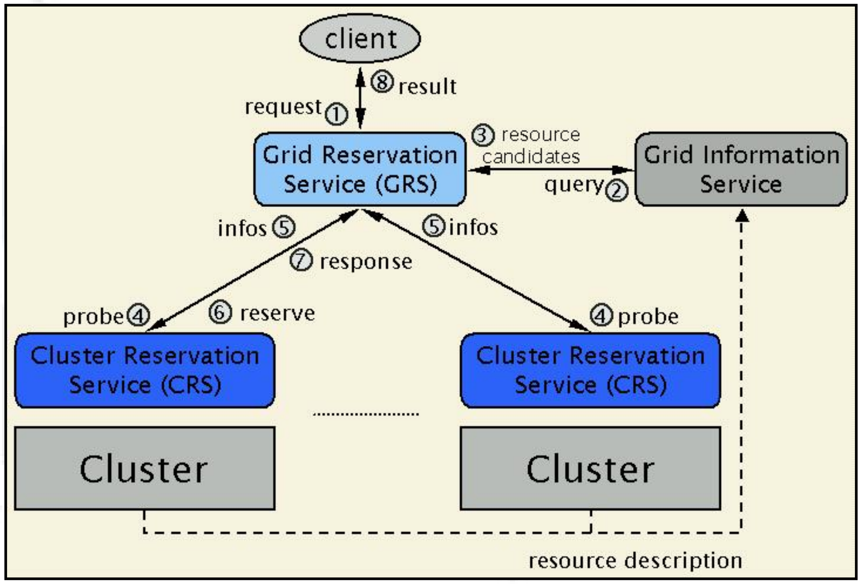


# Algorithm – Probe Phase Overview

- goal: minimize number of tries in case of success ***AND*** failure
- idea:
  - obtain additional information about system utilization from the CRSs
  - only consider candidates where the system utilization promises success

    ➔ reservation probability value (esr – estimated success rate)

**Note!** Mechanism can be used for arbitrary attributes (e.g. cost).



# Algorithm – Probe Phase ESR

- Domain for *esr* is $[0,1] \subset \mathbb{R}$.
- static: the longer the book-ahead period $t$ is the higher is the *esr* value

➔ $$esr(t,h) := 1 - e^{-\frac{t}{h}}$$

- history: determine average of idle processors $idle^{NP}$ for the requested time range using a \`calendar´ of logged utilization records

➔ $$esr(t,np) := \begin{cases} 1 & , 2np < idle^{NP}(t) \\ 2 + \dfrac{2np}{idle^{NP}(t)} & , \dfrac{idle^{NP}}{2} np \le idle^{NP}(t) \\ 0 & , elsewise \end{cases}$$

- load: approximate the time $T_{wkl}$ that is reached, when the current workload (running + idle jobs and existing reservations) is finished

➔ $$esr(t,T_{wkl}) := \begin{cases} 1 & , t \ge T_{wkl} \\ 0 & , t < T_{wkl} \end{cases}$$

# Algorithm – Reserve Phase Overview

- problem: many candidates
- goal: select `best´ candidate among the many
- idea: sort the possible candidates according the user‘s preferences
- preferences are prioritized, for example
  - 1st level: *end time*
  - 2nd level: *np* (number of processors)
  - 3rd level: *cost*
- any metric may be used within the preferences

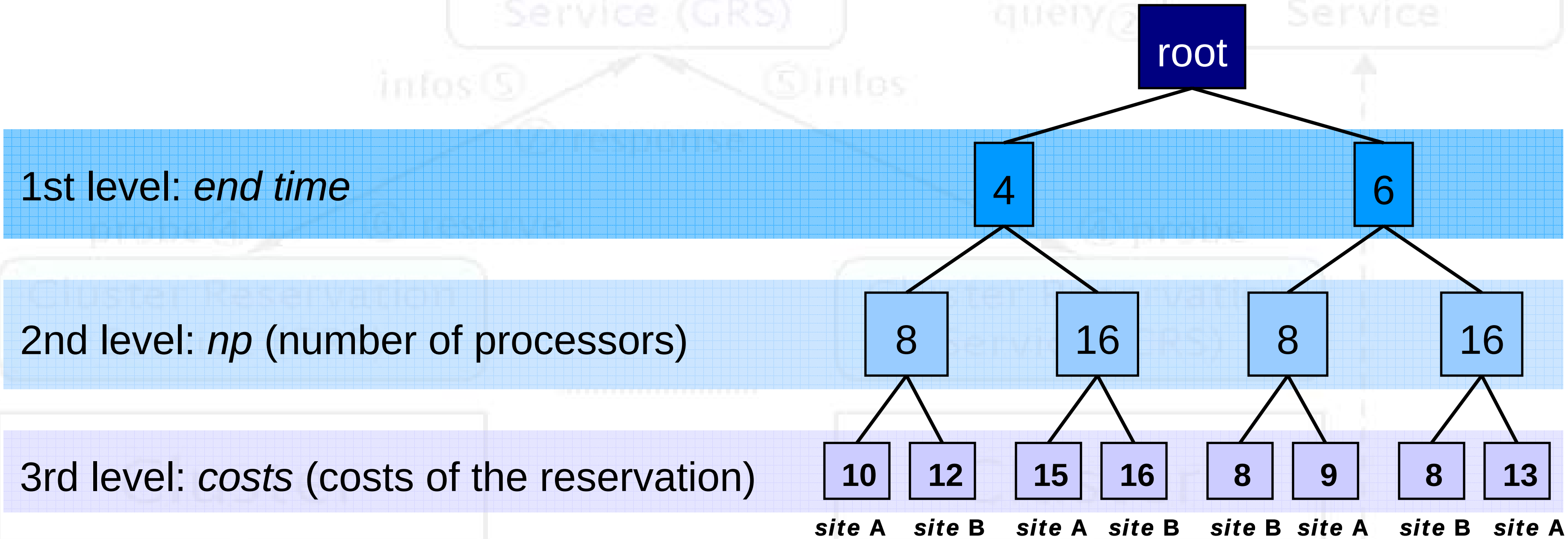
➔ GRS probes the CRSs to calculate the values of the metrics that are used in the preferences

# Algorithm – Reserve Phase Preferences Mgmt.

- idea: construct tree where the *i*-th level is sorted according to the *i*-th preference
- example: consider the following candidates

  { (site,start,end,np,costs) } = {

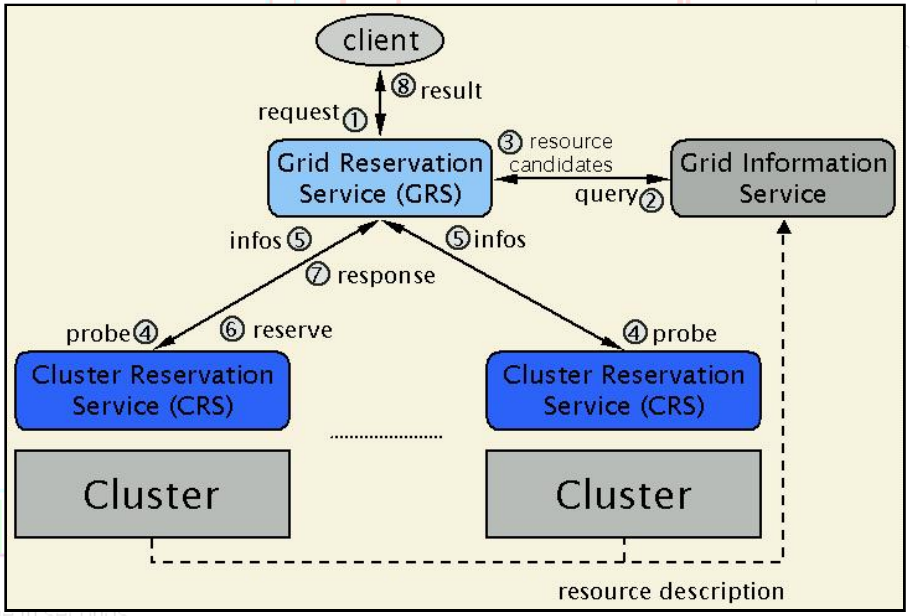
  (A,0,4,8,10), (A,1,4,16,15), (A,2,6,8,9), (A,3,6,16,13), (B,1,4,8,12 ), (B,2,4,16,16), (B,3,6,8,8), (B,4,6,16,8) }




- traverse tree in depth-first manner

# Evaluation

- do not use/simulate the Grid Information Service (GIS) *(only one cluster)*
- use a workload trace from a real system and simulate it with the Maui scheduler
  - ➔ subset (2k jobs) of the workload for the SDSC IBM Blue Horizon (144 8-way nodes)
- generate a reservation workload based on the subset of jobs
  - ➔ important parameters are book-ahead time, time range, processor range



# Evaluation – Selected Results

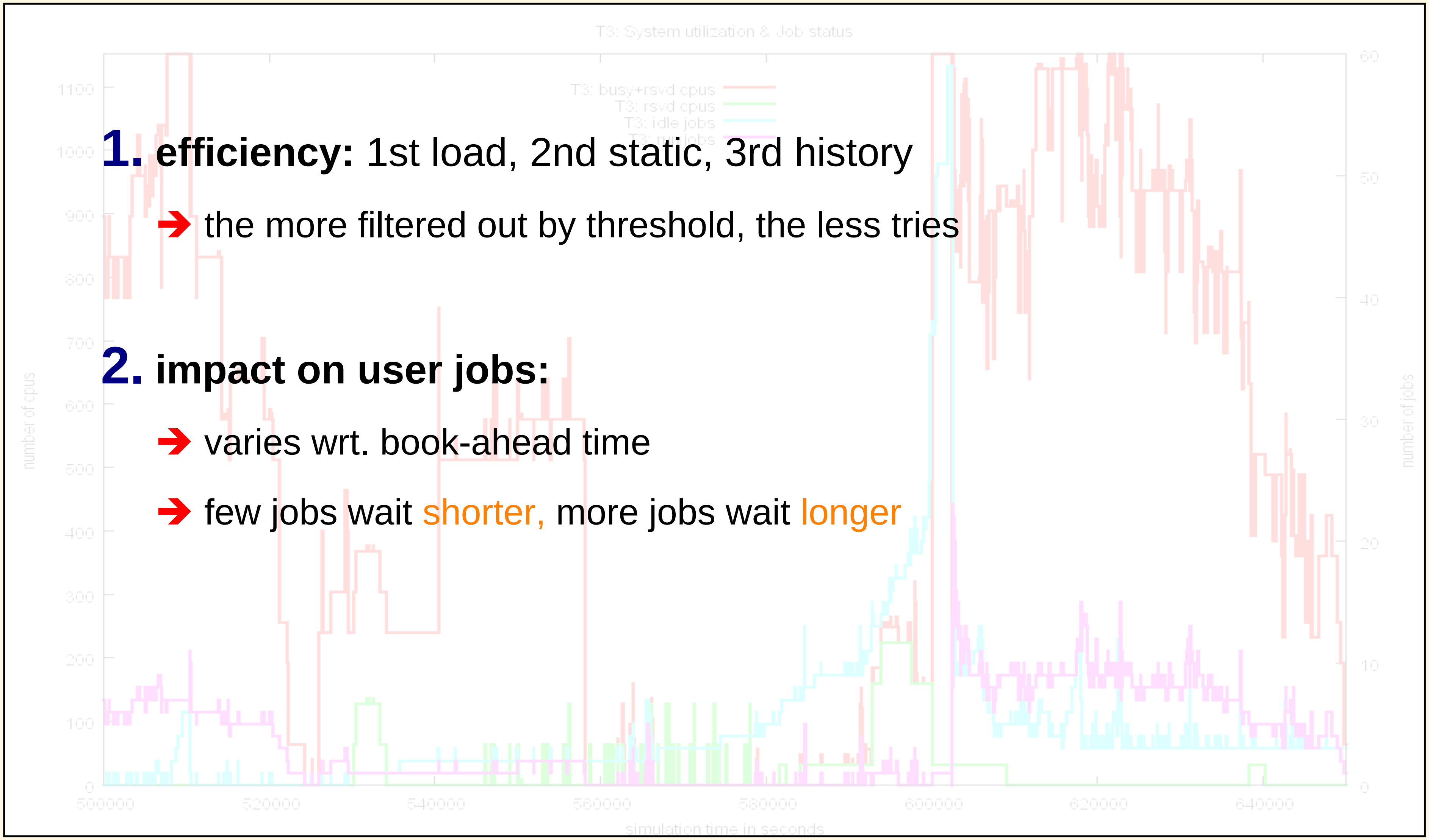
**1. efficiency:** 1st load, 2nd static, 3rd history

➔ the more filtered out by threshold, the less tries

**2. impact on user jobs:**

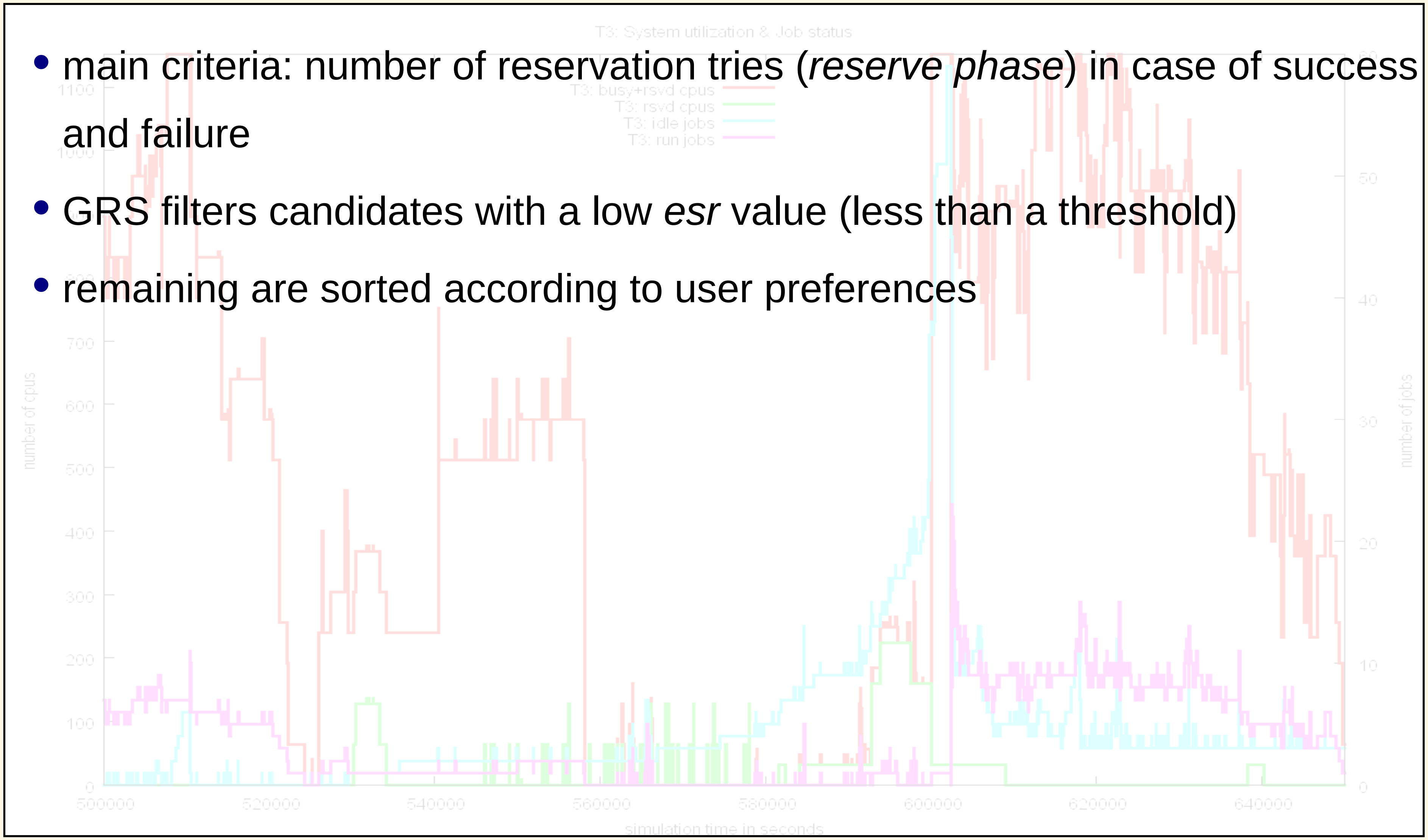
➔ varies wrt. book-ahead time

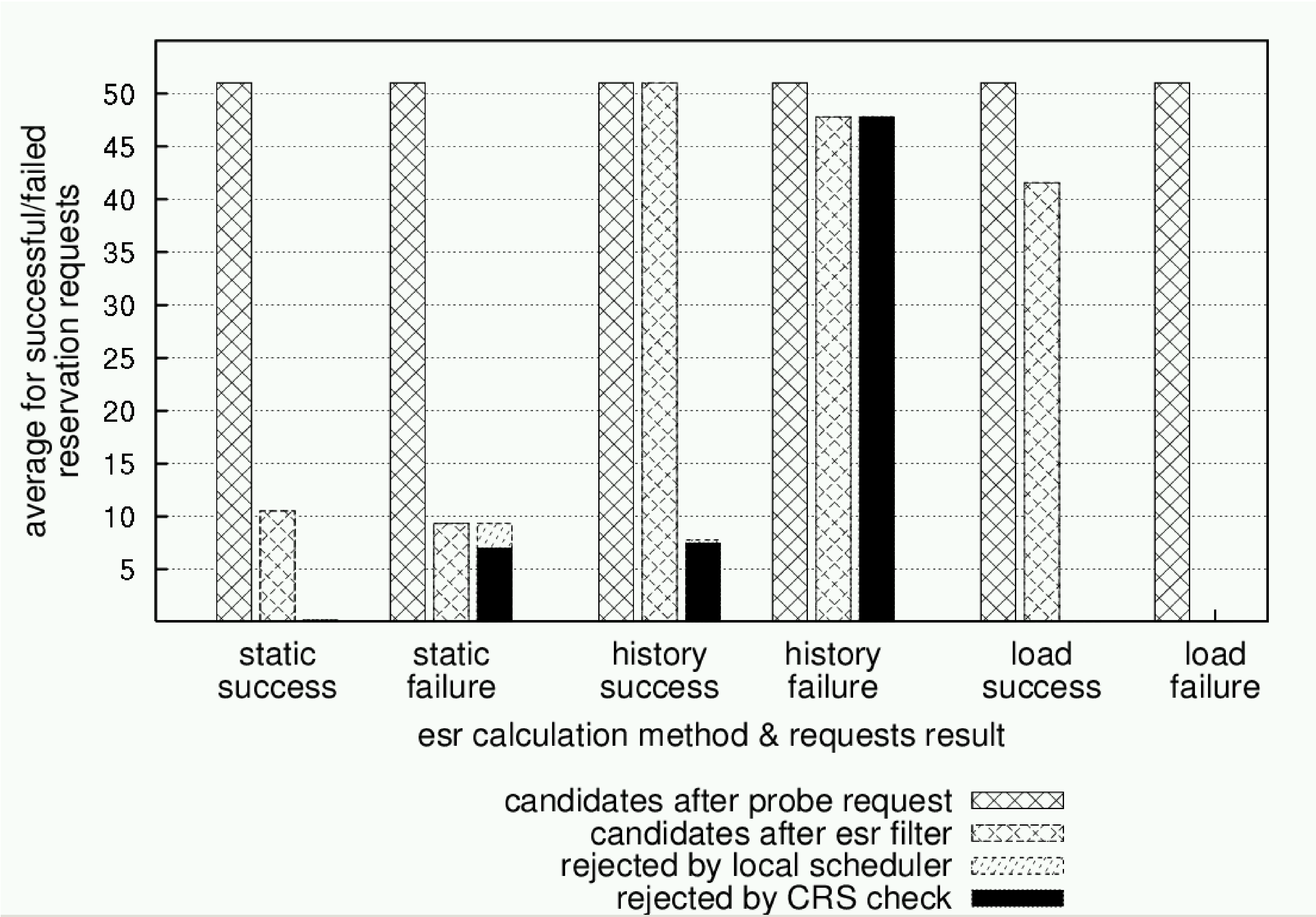
➔ few jobs wait shorter, more jobs wait longer



# Evaluation - Efficiency

- main criteria: number of reservation tries (*reserve phase*) in case of success and failure
- GRS filters candidates with a low *esr* value (less than a threshold)
- remaining are sorted according to user preferences

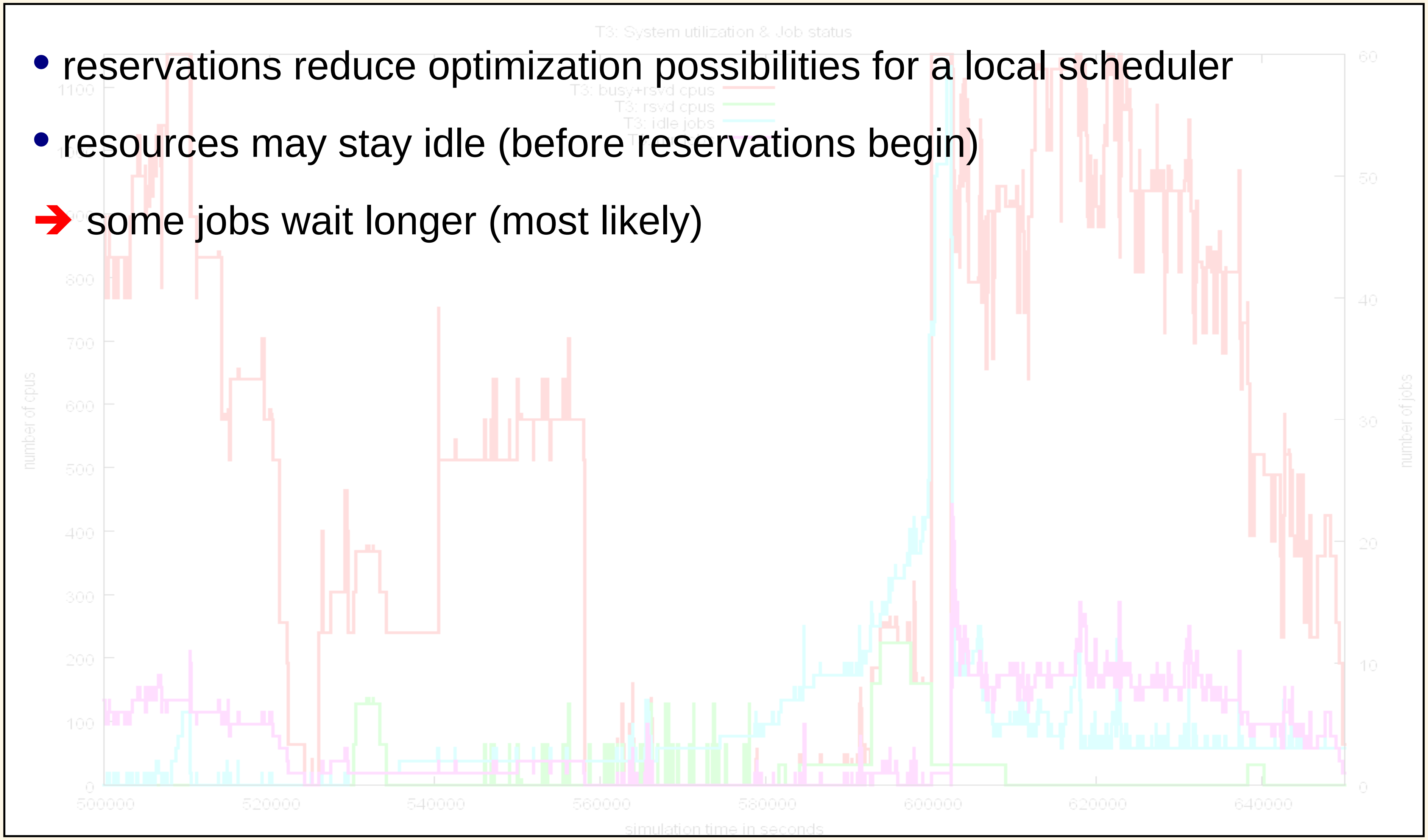


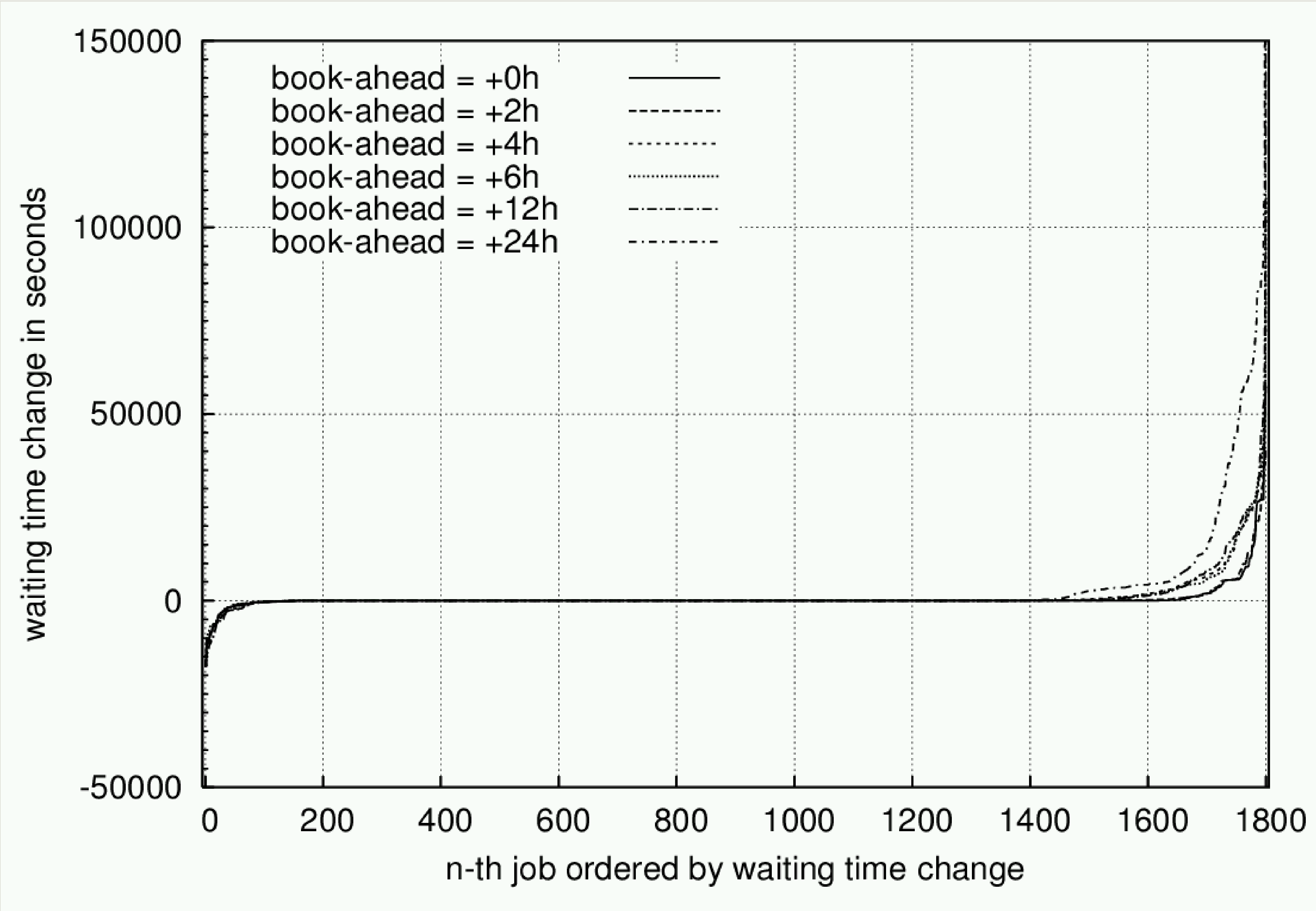

average for successful/failed reservation requests
50
45
40
35
30
25
20
15
10
5
static success
static failure
history success
history failure
load success
load failure
esr calculation method & requests result
candidates after probe request
candidates after esr filter
rejected by local scheduler
rejected by CRS check

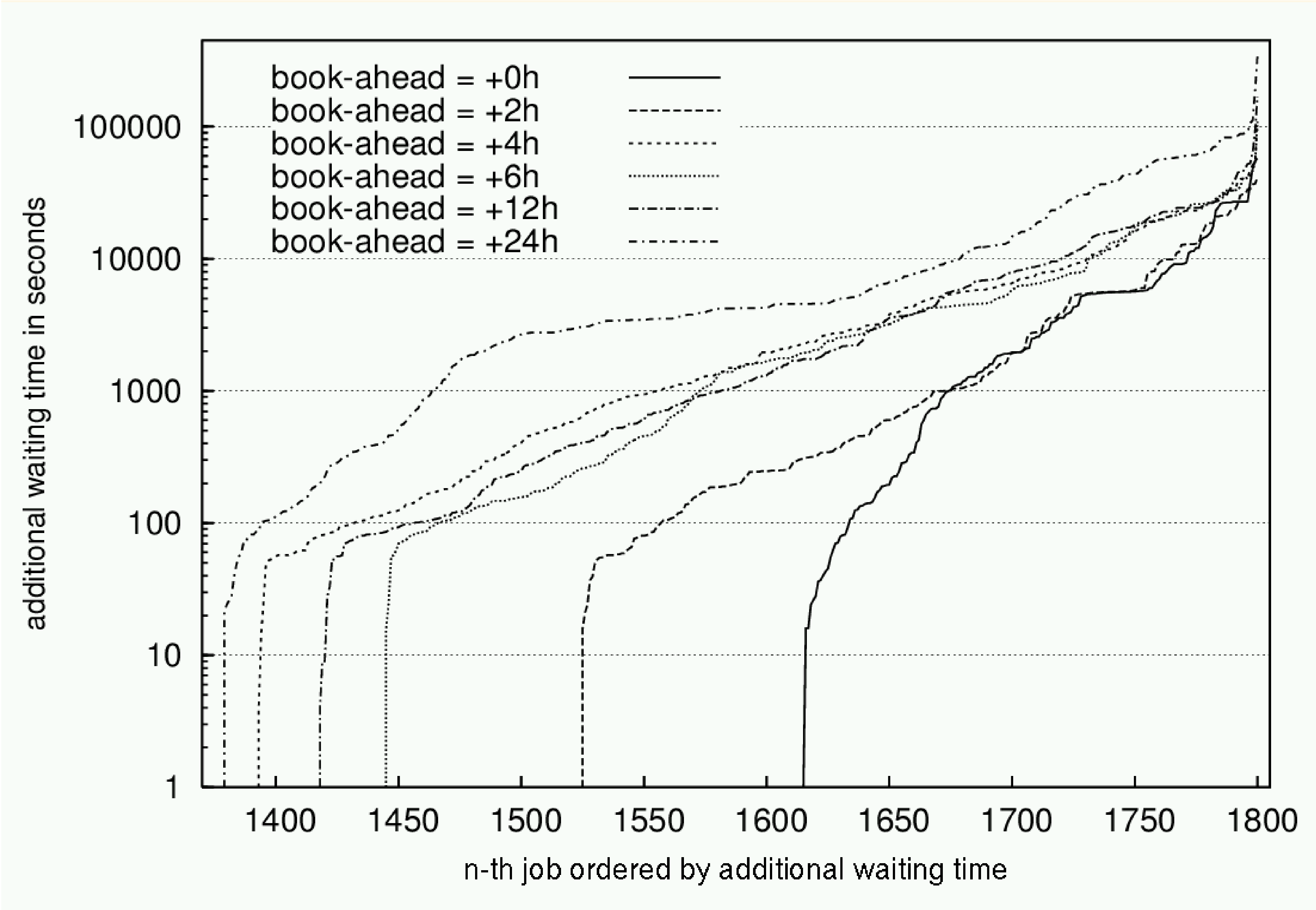
# Evaluation – Impact on User Jobs

- reservations reduce optimization possibilities for a local scheduler
- resources may stay idle (before reservations begin)

➔ some jobs wait longer (most likely)




book-ahead = +0h
book-ahead = +2h
book-ahead = +4h
book-ahead = +6h
book-ahead = +12h
book-ahead = +24h
150000
100000
50000
0
-50000
0
200
400
600
800
1000
1200
1400
1600
1800
waiting time change in seconds
n-th job ordered by waiting time change


book-ahead = +0h
book-ahead = +2h
book-ahead = +4h
book-ahead = +6h
book-ahead = +12h
book-ahead = +24h
100000
10000
1000
100
10
1
1400
1450
1500
1550
1600
1650
1700
1750
1800
additional waiting time in seconds
n-th job ordered by additional waiting time

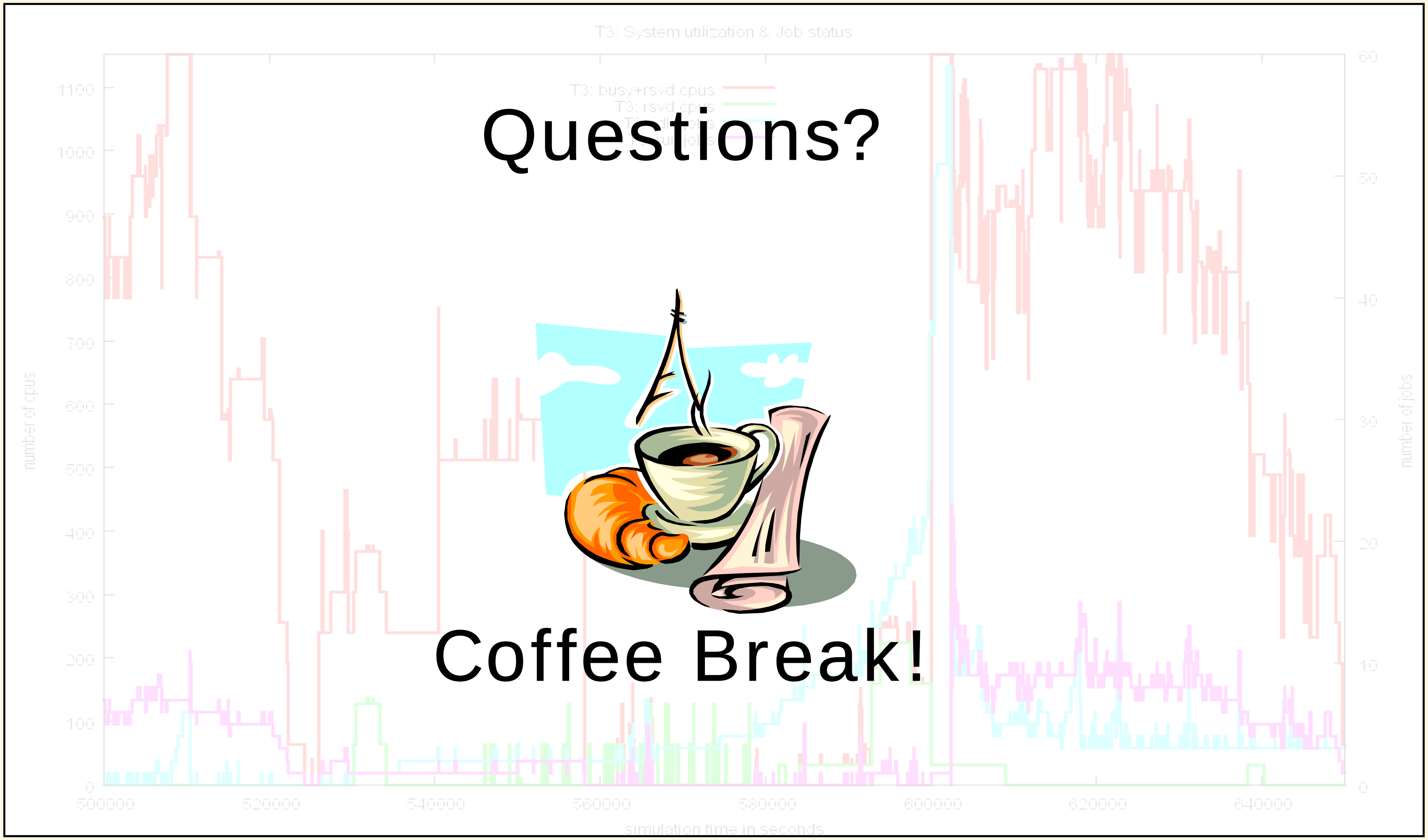
# Conclusion

- Reservation requests should be **elastic** to allow optimizations (site, start, #CPUs).
- Processing elastic reservation requests requires:
  - o **efficient** algorithms
  - o possibility to define **user policies**

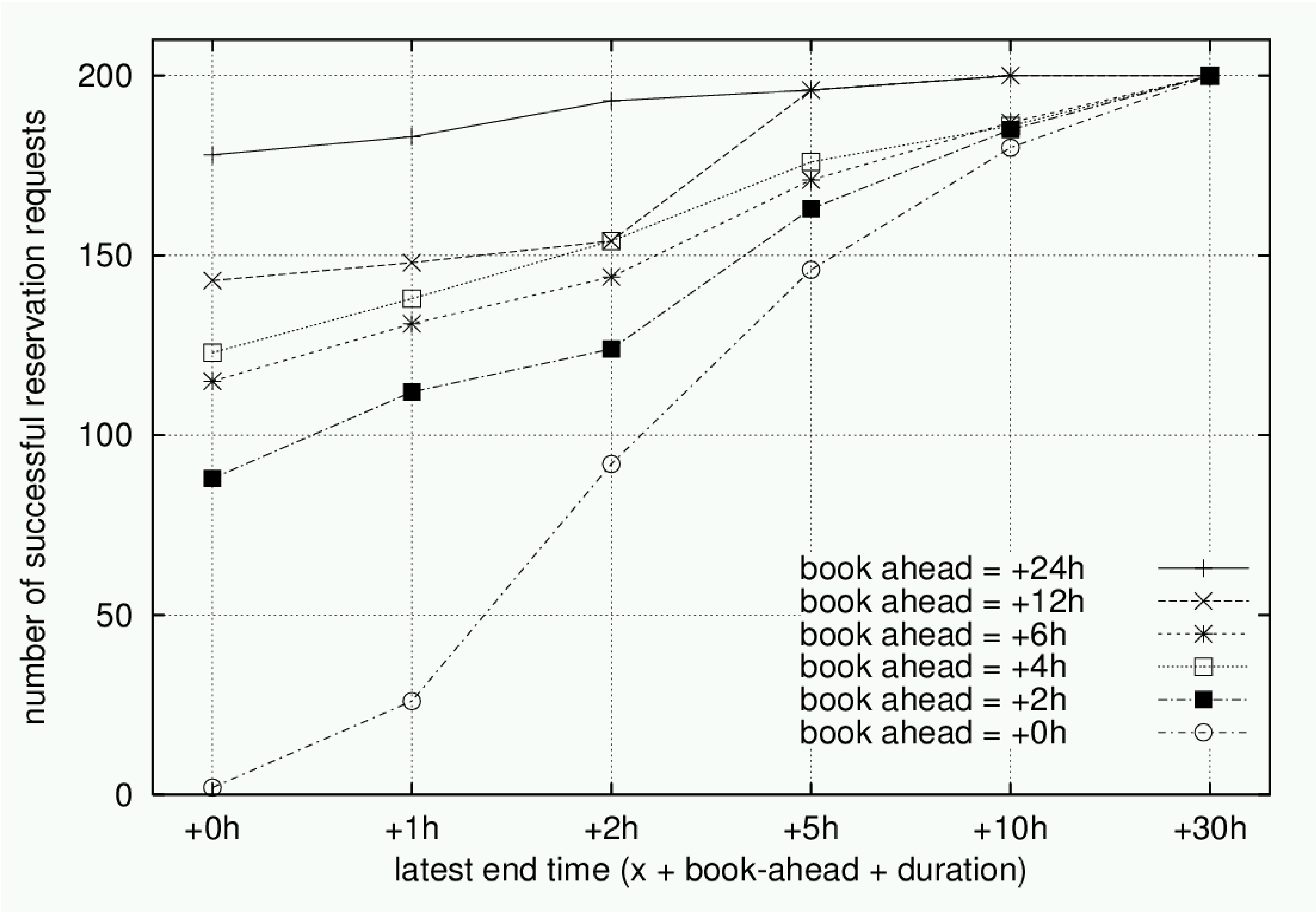
➔ Introduced a **probe** phase for calculating properties of reservation candidates:

- o extensible,
- o efficient &
- o user preferences

# Questions?




# Coffee Break!



number of successful reservation requests
200
150
100
50
0
+0h
+1h
+2h
+5h
+10h
+30h
latest end time (x + book-ahead + duration)
book ahead = +24h
book ahead = +12h
book ahead = +6h
book ahead = +4h
book ahead = +2h
book ahead = +0h