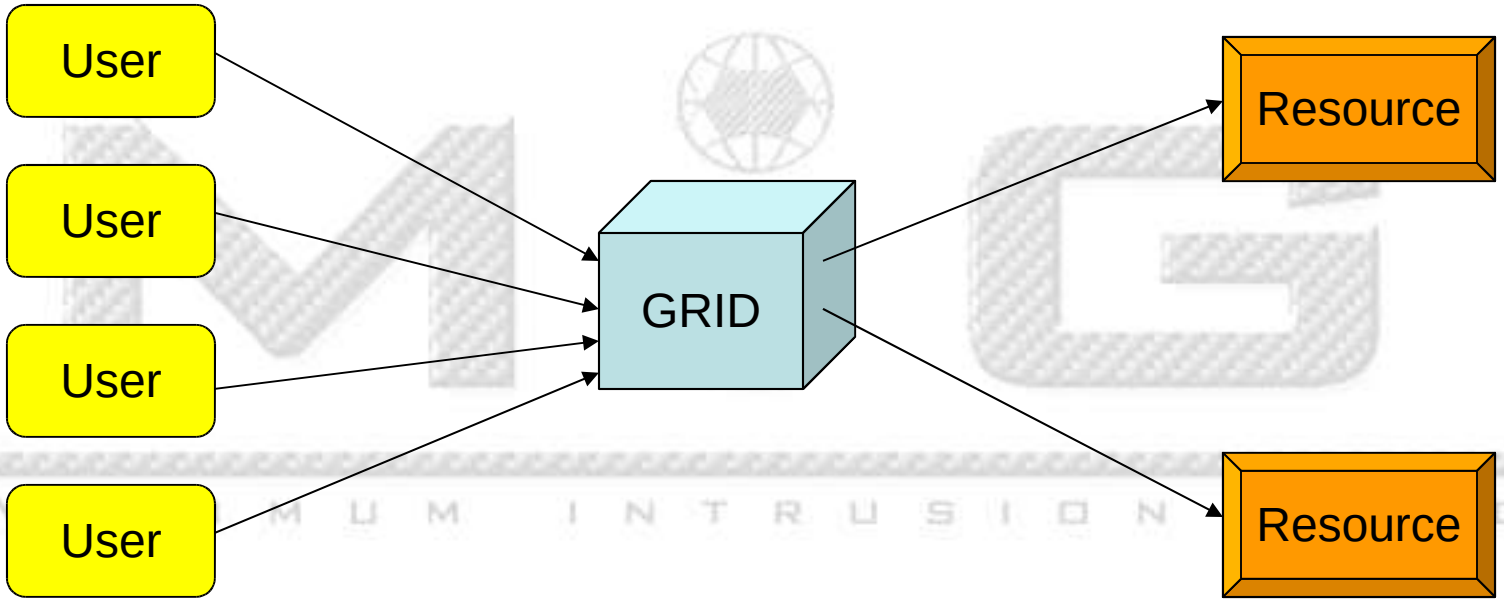
Minimum intrusion Grid

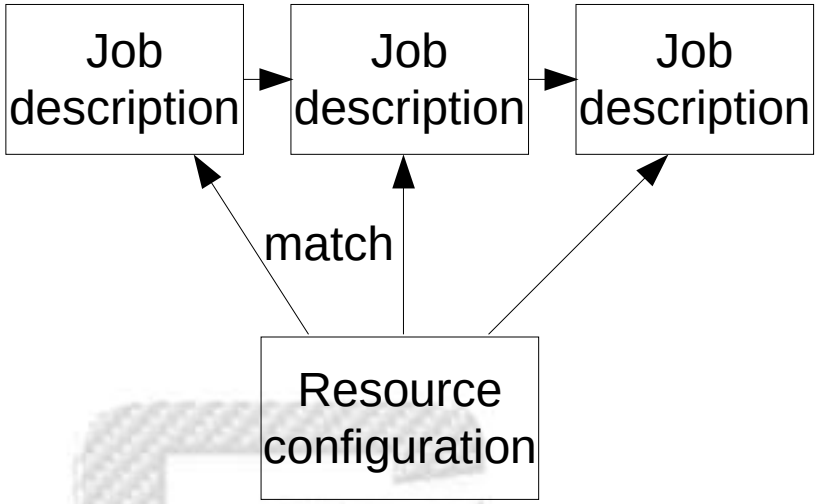
Economics and Load Balancing

Design and implementation

The Simple MiG Model

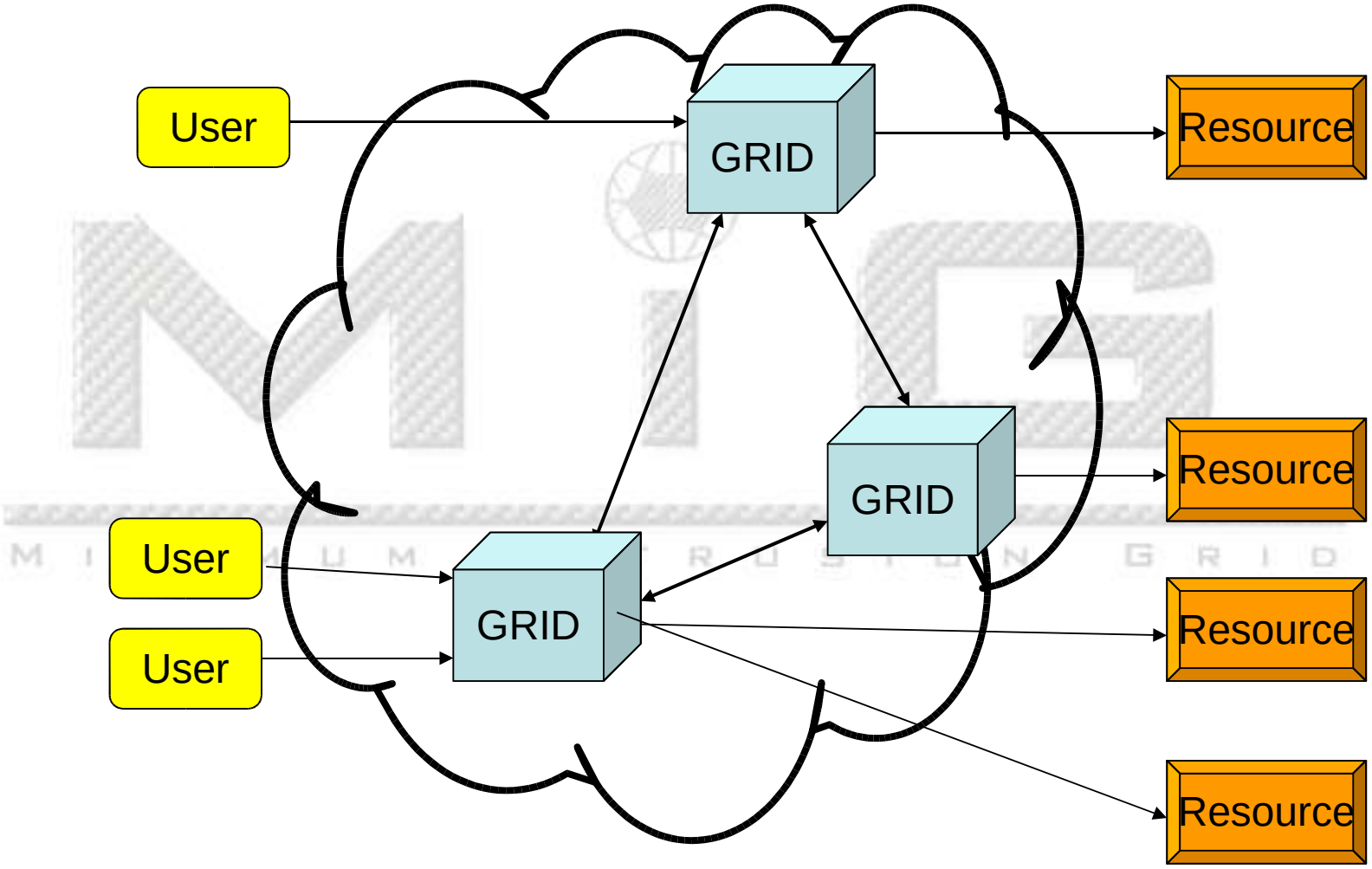


Simple Scheduling



- Simple model
  - Job queue
  - Local scheduling only
    - First fit, Best fit, Random, FIFO
    - Throughput (on-line algorithms)
  - Schedule when resources request job
    - CPUs, Nodes, Memory, Disk, REs
    - Price functions (experimental)
  - No actual economy
  - Fairness?

The Full MiG Model

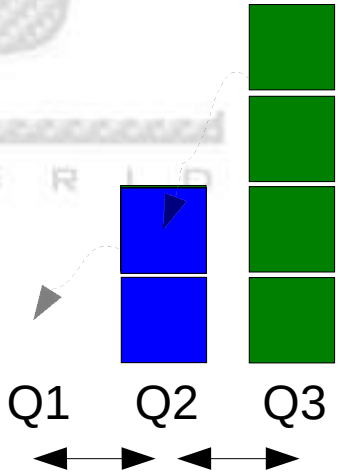


Full Model

- MiG Servers
  - Job queue
- Fault tolerance / Load balancing
  - Replication
  - Migration
- Economics
  - Pricing
  - Banking

Fault tolerance / Load balancing

- Fault tolerance
  - Replicate jobs and write to disk
  - Submit job: block until replicated
- Load balancing
  - Envy based balancing
  - Combined with pricing?
  - Migrate jobs that don't fit?
- Use same protocols



Economics

- Price properties:
  - Stable
    - price propagation, small local differences
  - Comparable
    - MiG units
  - Resource price, limited by owners (>minprice)
  - Job price limited by user (<maxprice)
  - “Fair” (user/resource/free market forces?)
  - Open economy:
    - deposit and withdraw funds from GRID

Stable, Fair Prices

- Market forces: supply and demand
  - Average “load”
  - Price propagation
- Actual price = minprice * load_multiply
  - minprice if no or low demand
  - grows with demand
  - as much as maxprice if plenty of jobs
- Price functions
  - job: delay
  - resource: time of day/week/month/year

Summary

- Market forces
  - Local prices
  - Based on demand (load)
  - Price propagation
  - Migration
  - Implicit price negotiation

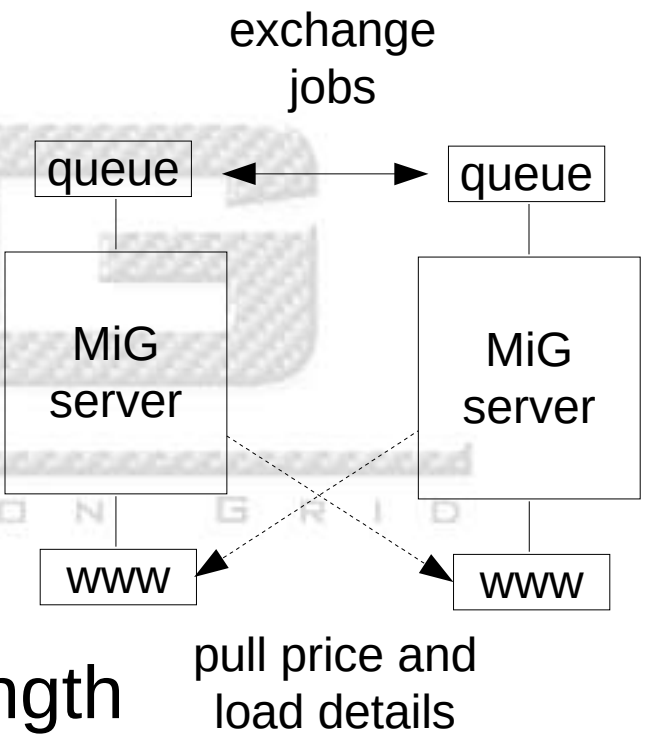
Server communication

- How and what to communicate
  - Job replication
  - Job migration
    - load/price balancing
  - Fail-over detection

- (py)cURL: HTTPS w. certificates
  - already in use, overhead (keep-alive)?

cURL

- Replication (unresolved)
  - push: cgi-scripts
- Price and load details
  - publish “conf” at local www
  - pull+parse remote “conf”
- Migration (unresolved)
  - based on price and queue length
  - push or pull “pickled” jobs
  - ... or steal replicated jobs if possible



Dynamic Pricing

- MiG unit (unresolved)
- Unit minprice in resource conf
- Job maxprice in mRSL
- Dynamic price
  - load multiplier
    - accept rate and target rate
- Price stability
  - price directed migration
  - cheap resources accept more jobs and vice versa

```
if load < target_load and load < last_load:
        decrease(mult)
elif load > target_load and  load > last_load:
        increase(mult)
```

Price Example

- Two resources
  - first fit
  - after warm-up
  - plenty of jobs
    - one max price: 500
  - load markers
  - stable actual prices
    - 498 to 503 (1%)
- Need more simulations!

Server status:
lo_load = 0.45
target_load = 0.75
queued = 22
fqdn = localhost
hi_load = 0.85

Resource reasonable.imada.sdu.dk:
load = 0.7
min_price = 80 ←
load_multiply = 6.235
cur_price = 498.8 ←
last_load = 0.7

Resource affordable.imada.sdu.dk:
load = 0.7
min_price = 100 ←
load_multiply = 4.985
cur_price = 498.5 ←
last_load = 0.7