Moulding Code: Distributed systems from sequential code

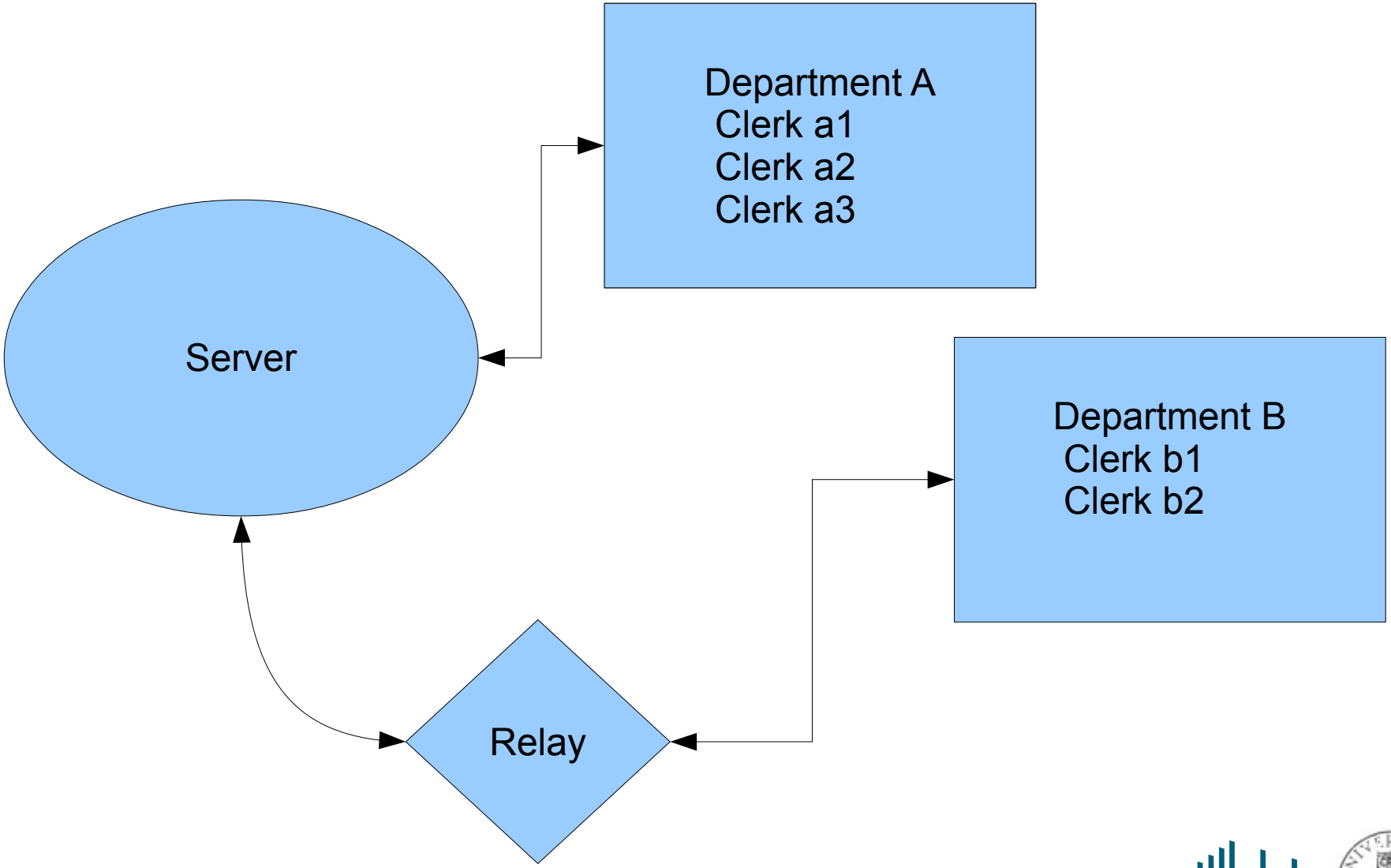
Magne Haveraaen

Bergen Language Design Laboratory (BLDL)
Department of Informatics, University of Bergen, Norway

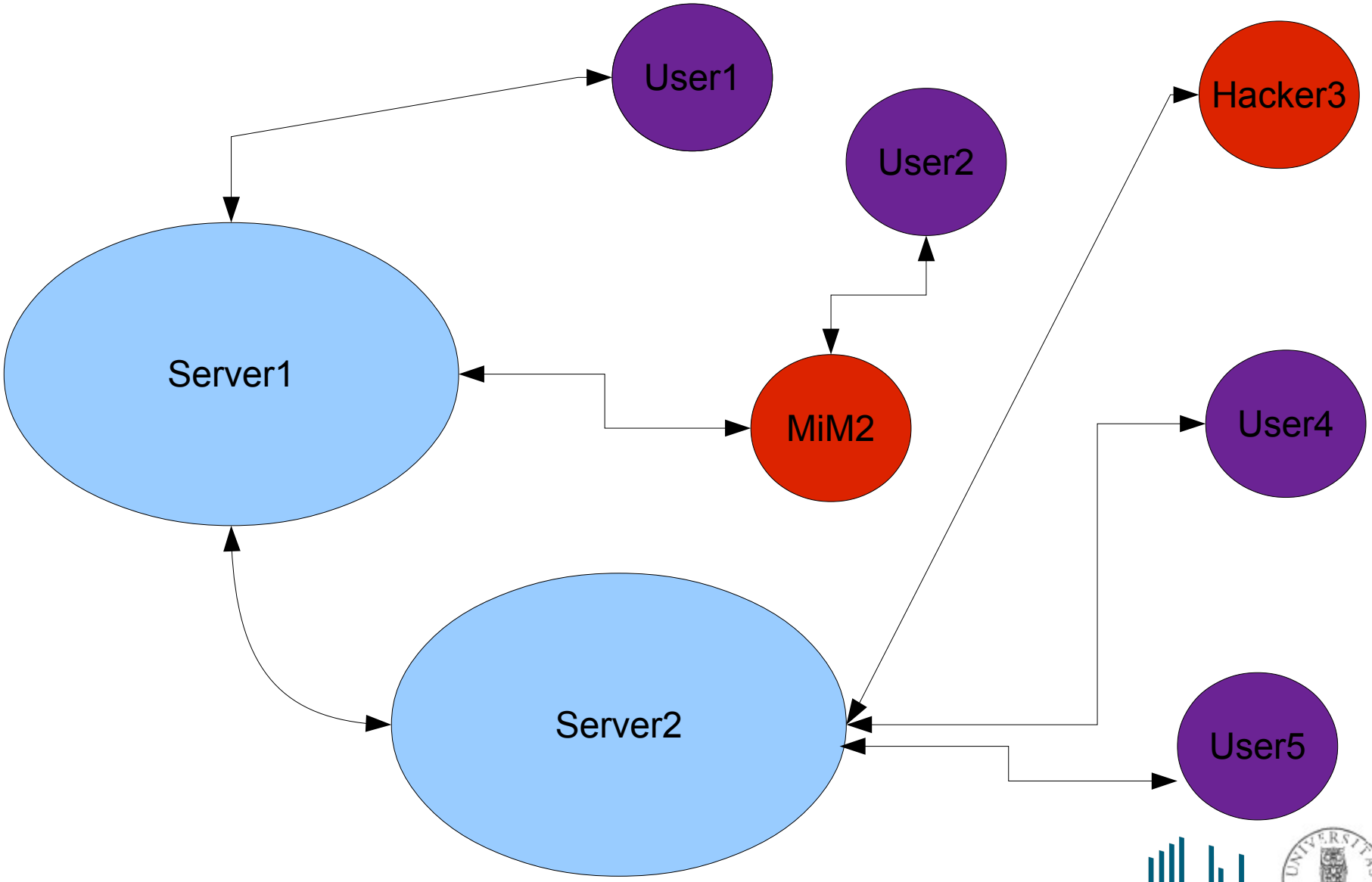
High Integrity Day

Bergen 2014-02-11

Distributed system architecture – Then



Distributed system architecture - Now



Traditional distributed systems development

- Each component developed “in isolation”
  - Server side, user side
  - Multi-threaded code
    - Interrupts
    - Remote procedure call
    - Callbacks
    - Shared variables
    - Synchronised regions
  - Security protocols
  - Secure communication
- Non-technical aspects
  - User interface: design and logic
  - Guiding of users in safe behaviour

Moulding code: develop sequential system

Sequential systems are hard enough to develop

- Define each transaction as a separate method.
- Access the data base synchronously.

Benefits

- Simple control flow.
- Reasoning about correctness (inside programmers head)
  - Preconditions
  - API specifications / postconditions
  - Algorithms for each method
- Unit testing for correctness: applies at every level

Moulding code: split into server and user sides

Tool support for reorganising the code:

- Split each transactional method in two parts:
  - Server side
    - Automatically insert validation of input data formats
  - User side
    - Automatically insert validation of input data formats
  - Communication code between server and user sides

    Automatically generated
    - Use encrypted connections
    - Choice of security protocols

Moulding code: server side

Tool support for reorganising the code:

- Collate server side transaction code into one system
- Make code reactive
  - Multi-threaded
  - Interrupt-driven
- Duplicate services for higher throughput

Moulding code: user side

Tool support for reorganising the code:

- Define UI for each user side transaction method
  - UI logic based on transaction's IO requirements
  - UI look & feel added separately and connected to the logic
- Collate user side transaction code into one app
- Make code reactive
  - Multi-threaded
  - Interrupt-driven

  Allows asynchronous send and receive of data with UI

Conclusion

- Traditional distributed systems development
  - Independent components
  - Multi-threaded
  - Asynchronous access
  - UI hand coded as reactive system

  Very difficult to get correct
- Vision of mouldable software development
  - Develop simple synchronous transaction code
  - Mould code into reactive server side and user side
    - Automatically include security features
  - Add UI logic and look and feel separately

  A possibility for high integrity software