Functional Programming^{XP}

The Industrial Experience

karol.ostrovsky@gmail.com

OSTROVSKY
RESEARCH INSTITUTE

Karol Ostrovský

- M.Sc. – *Comenius University, Bratislava*
- Ph.D. – *Chalmers*
- Post-doc – *Chalmers*
- System Designer – *Dfind IT*
  - On assignment for Ericsson
- Design Architect – *Ostrovsky Research Institute*
  - On assignment for Ericsson

The Chalmers Years

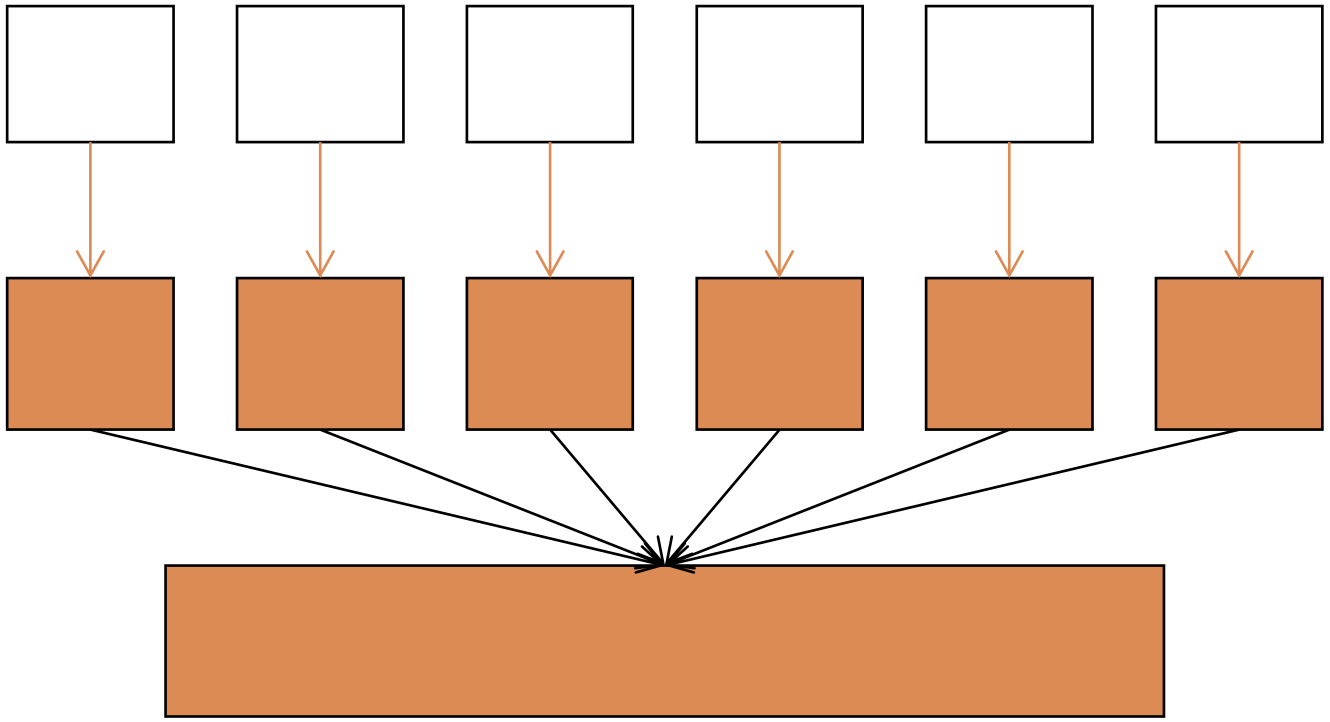
- Research in static analysis of concurrent programming languages
  - Type systems
  - Protocol analysis
- Main course responsible
  - Concurrent Programming Course – was TDA381
  - Developed the course between 2005 and 2010

The Language & Paradigm Nerd

- Basic
- Pascal
- C/C++
- Scheme
- SmallTalk
- Java
- JR (MPD)
- Haskell
- Erlang
- Prolog
- Python
- Ocaml
- LaTeX
- VAX assembler
- Trilogy
- Ada
- Agda
- ATL
- My own languages
- …

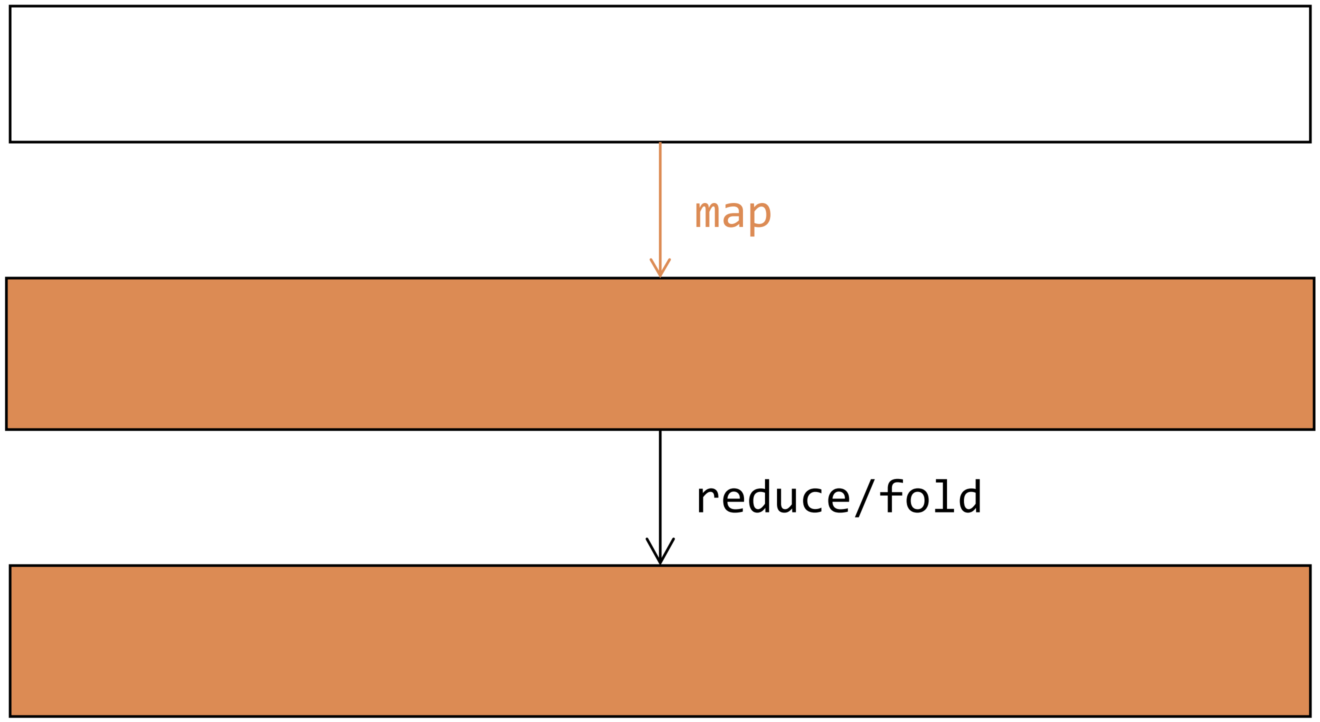
What is Programming?

- Manipulation of Structures



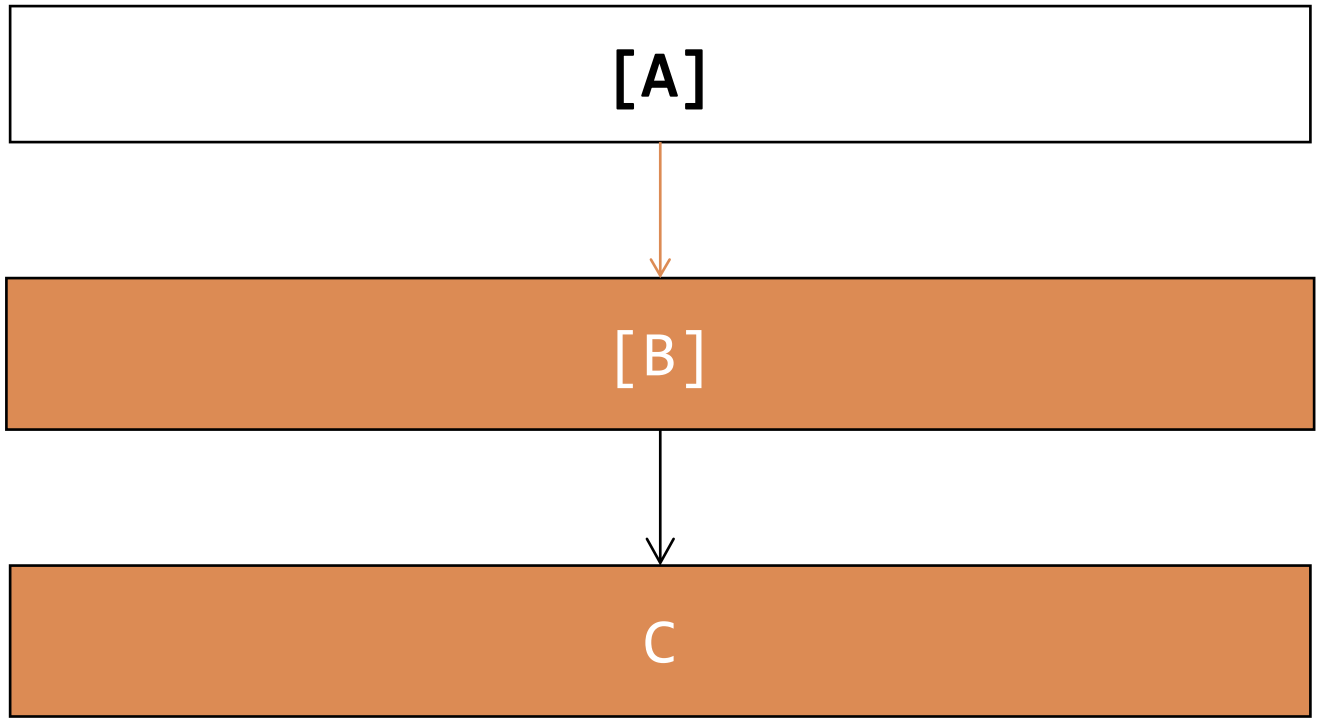
Compositions

- Functions

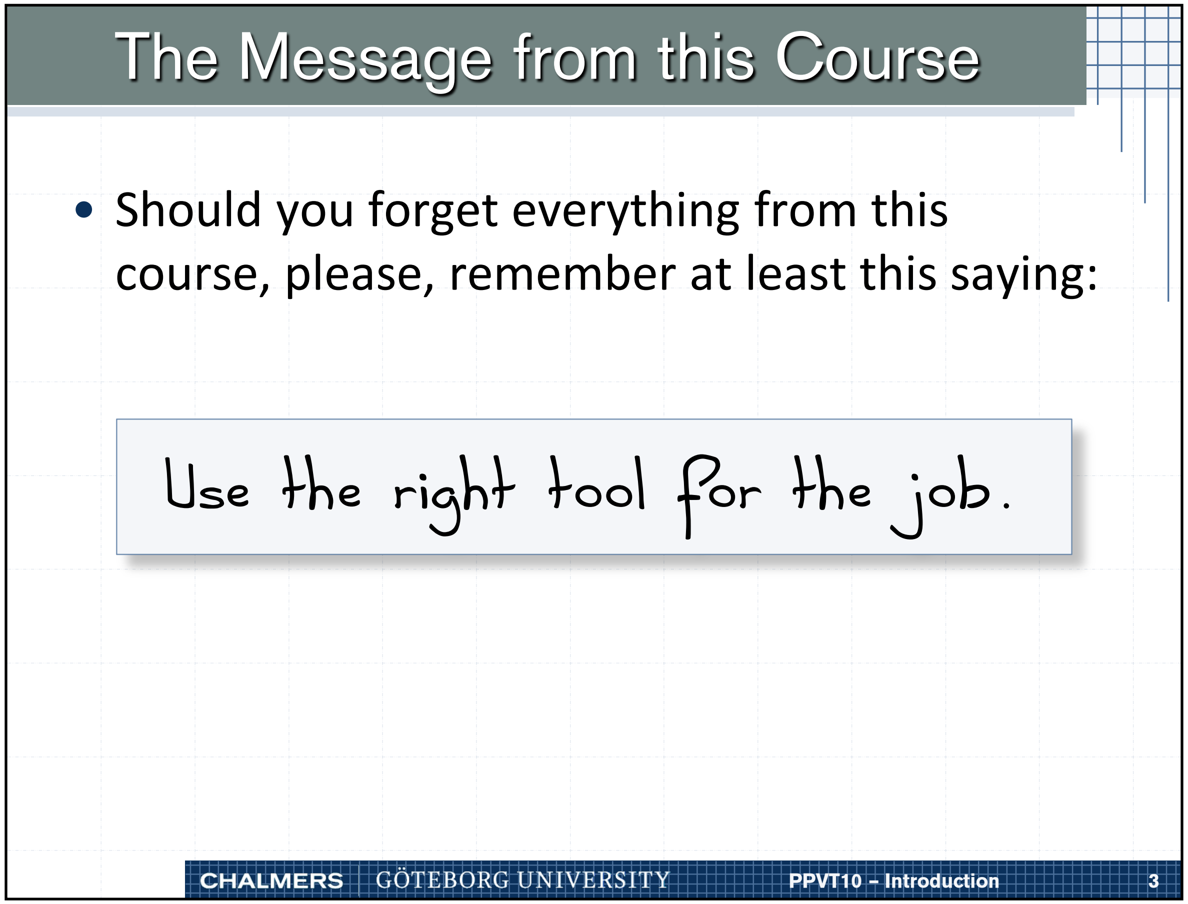


Structures

- Types



My Favourite Slide



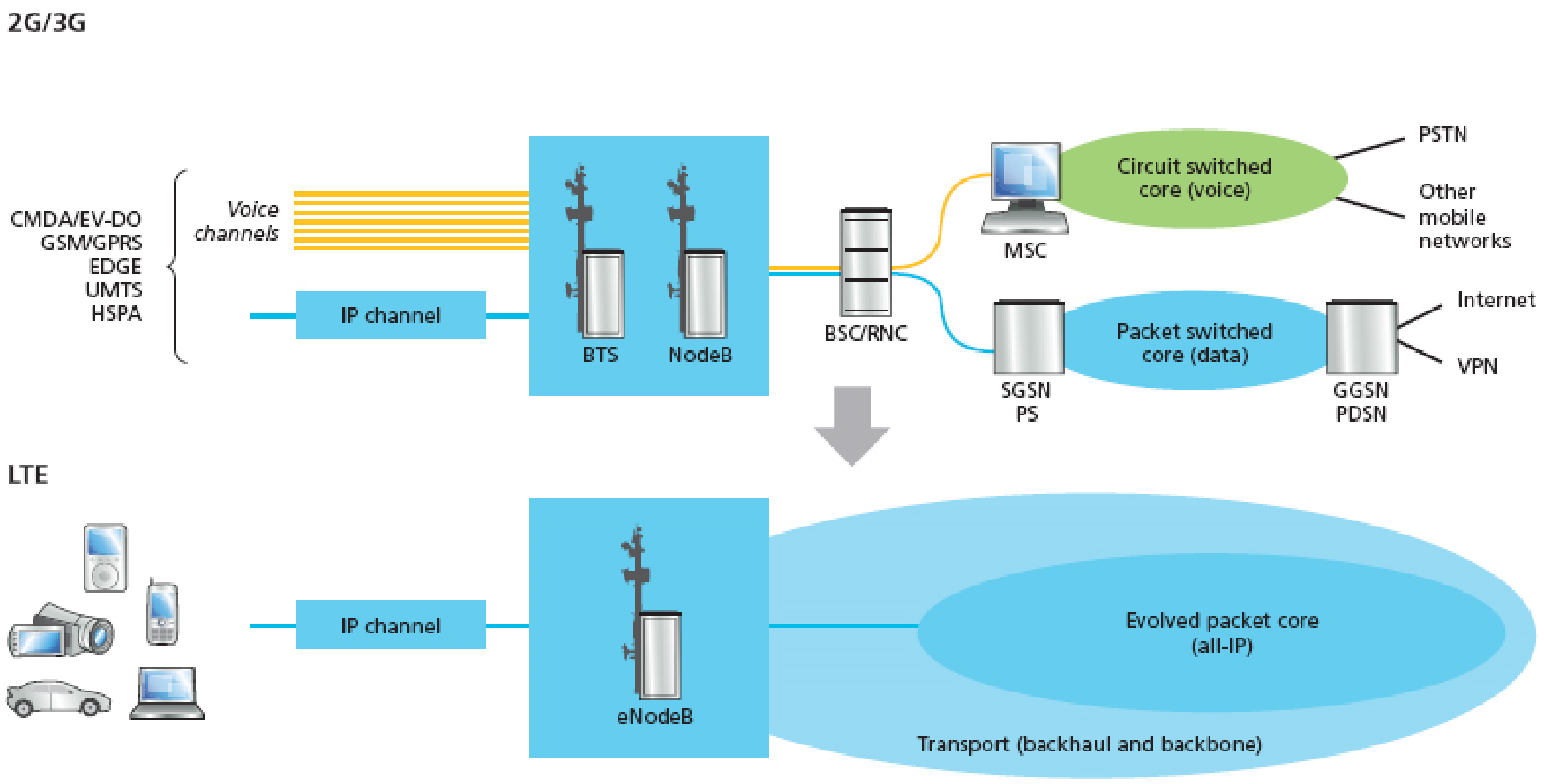
The Message from this Course

- Should you forget everything from this course, please, remember at least this saying:

Use the right tool for the job.

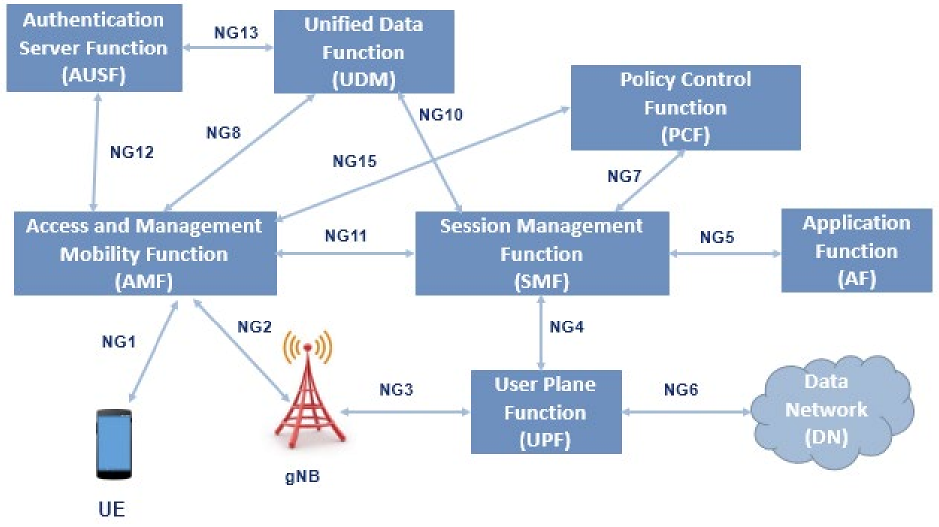
CHALMERS | GÖTEBORG UNIVERSITY PPVT10 – Introduction 3

Mobile Telecom Network



5G

- Soup of buzzwords
  - Container-based
  - Cloud-native
  - State-less
  - REST-API
  - …


Authentication Server Function (AUSF)
NG13
Unified Data Function (UDM)
Policy Control Function (PCF)
NG12
NG8
NG10
NG15
NG7
Access and Management Mobility Function (AMF)
NG11
Session Management Function (SMF)
NG5
Application Function (AF)
NG1
NG2
NG4
NG3
User Plane Function (UPF)
NG6
Data Network (DN)
gNB
UE

Mobile Telecom Standards

- Interoperability is essential
- The Internet Engineering Task Force
  - Develops and promotes voluntary Internet standards
  - Request for Comments (RFC)
- 3rd Generation Partnership Project (3GPP)
  - Defines telecom standards

Packet Core Network

- Ericsson Mobility Management
    - SGSN – Servicing GPRS Support Node (2G/3G)
    - MME – Mobility Management Entity (4G)
    - AMF – Access and Mobility Management Fun. (5G)
    - Control signalling
    - Admission control, Authentication
    - Mobility, roaming
    - Payload transport (not in 5G, 4G or 3GDT)

SGSN-MME MkVI

- 3 sub-racks
- 21 blades (2+19)
- 2 core PowerPC
- ~114 simultaneously running processes
- Backplane: 1Gbps
- Capacity: 3MSAU

SGSN-MME MkVIII

- 3 sub-racks
- 14 blades (4+12)
- 6 core SMT Intel x86
- ~432 simultaneously running processes
- Backplane: 1Gbps
- Capacity: 18MSAU

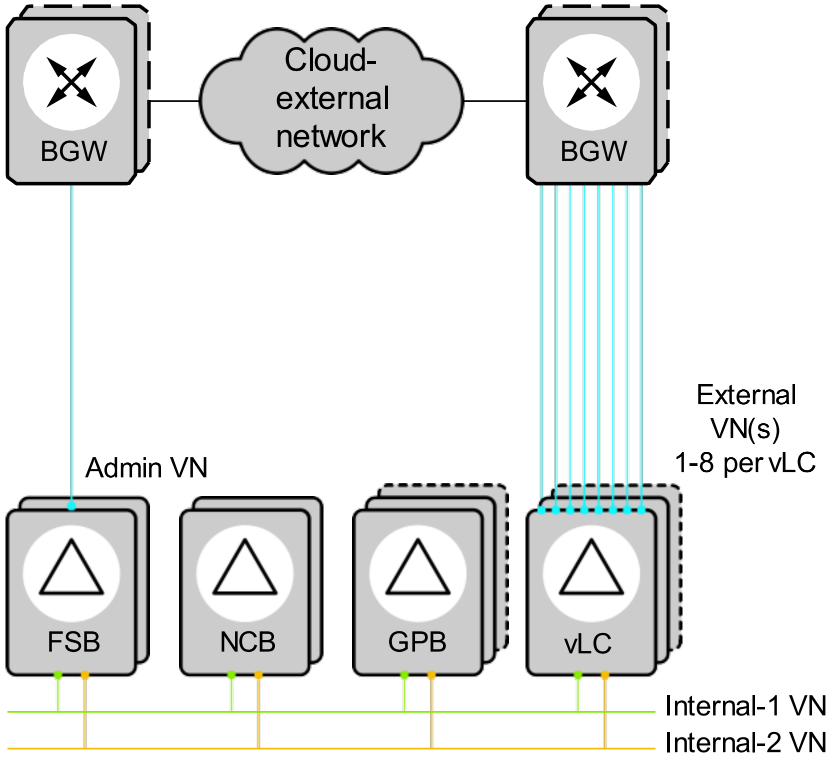
OSTROVSKY RESEARCH INSTITUTE

SGSN-MME MkX

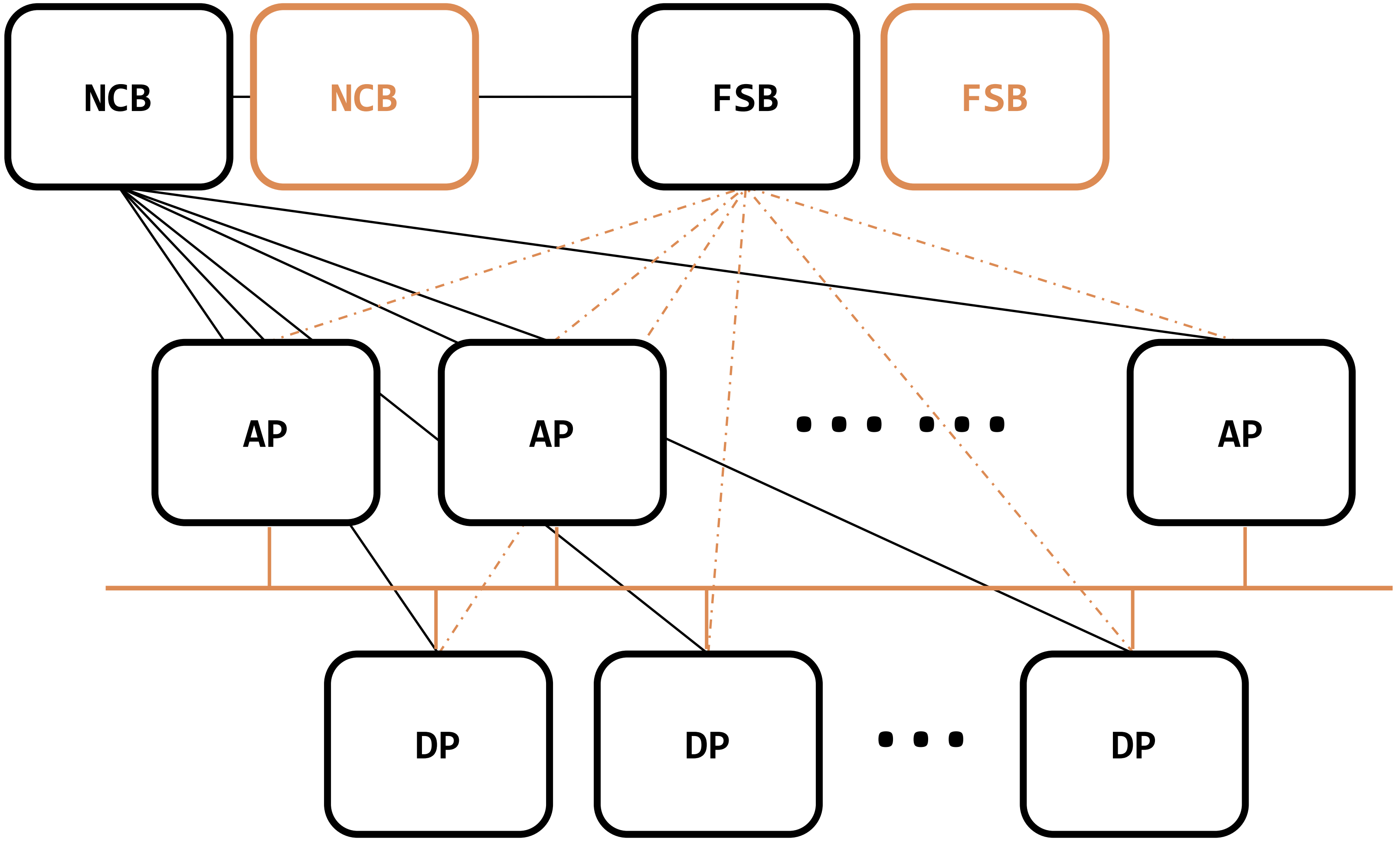
- 3 sub-racks
- 14 blades (2+12)
- 10 core SMT Intel x86
- ~720 simultaneously running processes
- Backplane: 10Gbps
- Capacity: 36MSAU

SGSN-MME virtualised

- Mirrors the logical components of the native HW design
- Flexible deployment
  - Hypervisors
  - Cloud Management



SGSN-MME – Architecture Sketch



SGSN-MME – Use The Right Tool

- Control Plane
  - Erlang
    - concurrency
    - distribution
    - fault-tolerance
  - DSL: frameworks for protocol implementation
- User Plane
  - C: time-critical packet forwarding

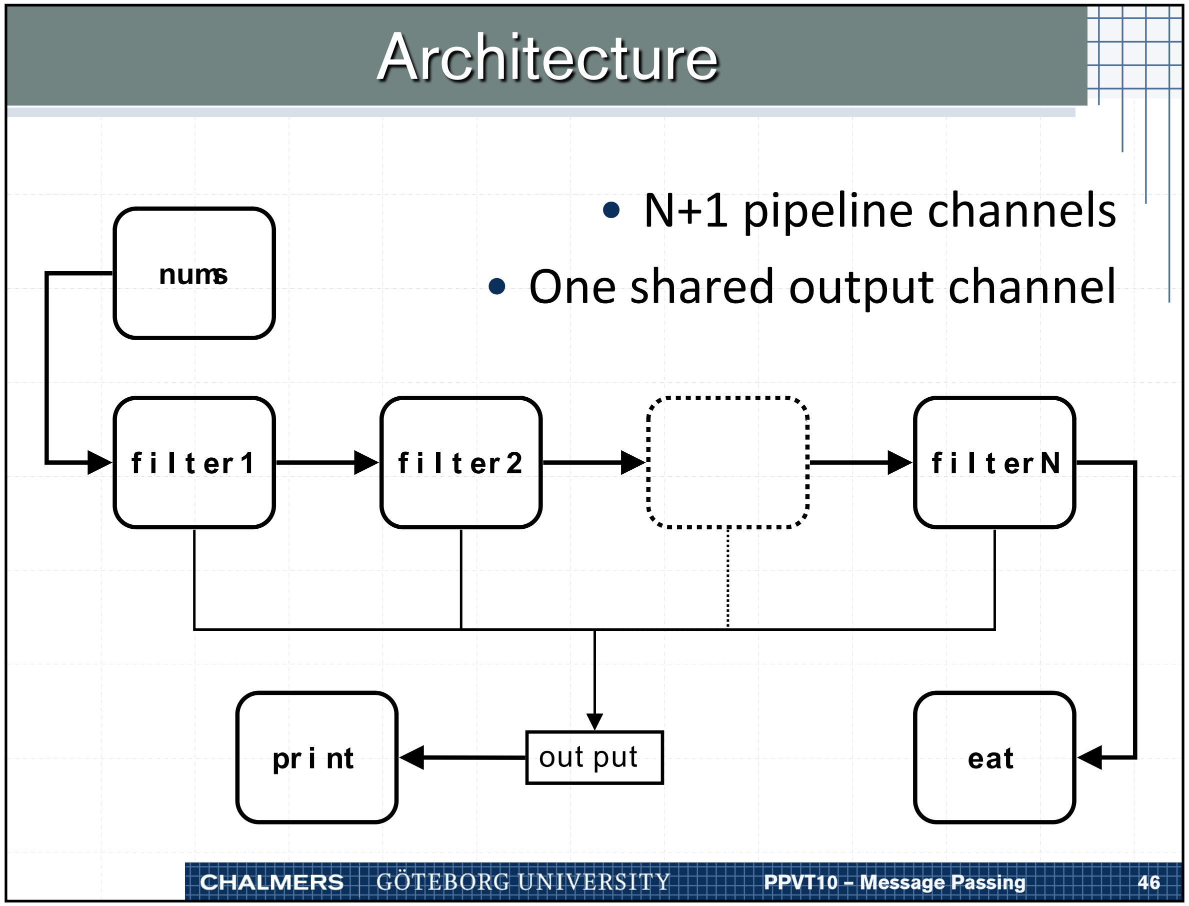
The Functional Advantage

- Protocol Programming
  - 3GPP standards
  - Domain experts not software engineers
- DSL
  - A "library" of abstractions
    - Possible in any language
    - Often easier in a functional language
  - A set of combinator "glues"
    - Considerably more powerful in a functional language

Typical Concurrency Patterns 1

- Barrier
  - Multi-stage barrier
    - Start-up of each AP boards internally
    - Middleware synchronisation across all boards
    - In service software upgrade
  - Value-carrying negotiable
    - One value producer
    - Multiple subscribers

Typical Concurrency Patterns 2



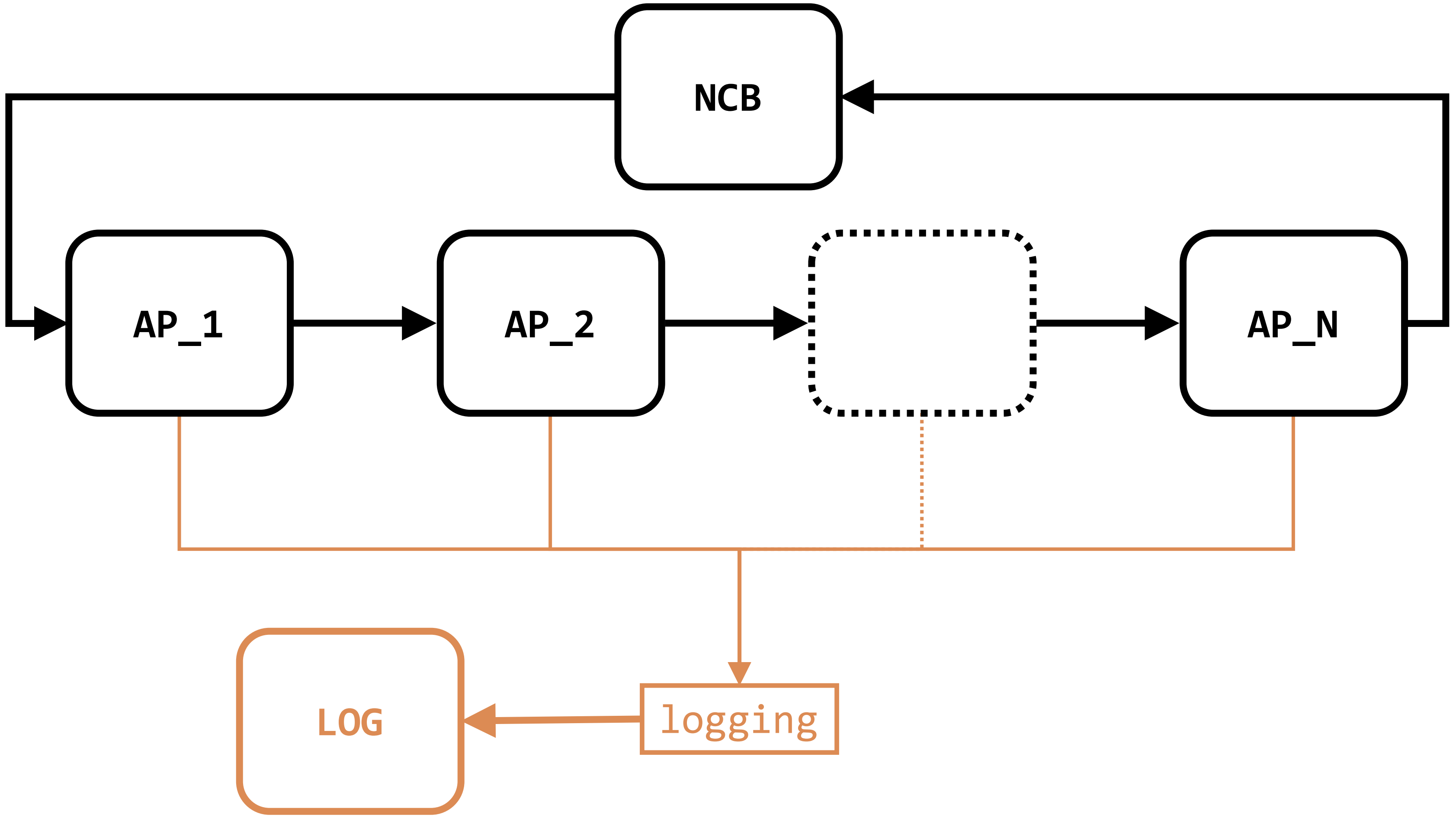
Architecture

- N+1 pipeline channels
- One shared output channel

CHALMERS GÖTEBORG UNIVERSITY PPVT10 – Message Passing 46

Sieve of Eratosthenes

Pipeline of Processes



Typical Concurrency Patterns 3

- One mobile – one process (replicated worker)
  - Isolation
  - Synchronisation only with resources
- Central resources
  - Resource allocator
  - Master/controller – slave/worker
  - Transaction handler
  - (Parallel and Distributed) Iterator

Distribution

- One mobile – one process
  - Evenly distribute all phones over all blades
  - Replicate data for fault-tolerance
- Central resources
  - Run on the controller-blade
  - Replicate to all the worker-blades
  - Can we survive without a controller?

Fault-tolerance

- SGSN-MME requirement: 99.999% availability
- Hardware
  - Faulty blades are automatically taken out of service
  - Mobile phones redistributed
- Software
  - Fail fast – offensive programming
  - Recovery strategy

Fault-tolerance – Software

- Phone process crash should never affect others
    - Automatic memory handling
    - Process monitoring
- Recovery Strategy – Escalation Hierarchy
    - Restart the phone process
    - Restart one blade
    - Restart the whole node

FP Patterns – Monads

- Good
  - Keeps pure and side-effecting computations apart
    - Good separation of concerns
    - Improved compositionality
    - Possible performance gain
  - Gather writes together and write to DB once – amortise the cost of transactions:
    - 1 item write costs 10
    - 10 items write is not 100 but only 20!

FP Patterns – Monads

- Bad
  - In rapid prototyping it can present a big hurdle to jump over
  - So, it is good that Erlang does not have static types
  - Lazy evaluation is more complicated in the presence of side-effects especially inter-process communication

OO-Design Patterns

- Factory method
  - Improve memory sharing
- Object pool
  - Bounded parallelisation of algorithms – thread pool
  - Overload protection
- Iterator
  - Perform operations on all phones

What they do not teach you

- Software lives long
  - Especially telecom systems (decades)
  - Banking systems live even longer (think Cobol)
- People change
- Organisations change
- Hardware changes
- Requirements change
- Documentation often does not change

Software Maintenance

- The developer's challenge
  - Write simple (readable) and efficient code:
    1. Write a straightforward and working solution first
    2. Optimise later (or even better skip this step)
- Think smart but do not over-optimise
  - Optimisations complicate maintenance
- The code is often the only reliable document
  - Types can be very good documentation

Synthesis and Analysis

- Emphasis on synthesis in education
  - Software development from scratch
- Industrial systems often have a legacy
  - Software development by further iteration
    - Refactoring
    - Code review
    - Software maintenance
  - Need for both analytical and synthesizing thinking

Synthesis and Analysis

- Roughly 30% of manpower is spent on testing
  - Analytical work
  - Do you like to break a system?
- But testing can also be “synthesizing”
  - Testing frameworks
    - QuickCheck
    - SGSN-MME has its own
  - Formally prove the system correctness?

Erlang in Practice – Pros

- Well suited for
  - Control handling of telecom traffic
  - Application layer (OSI model) applications
    - Web servers, etc.
  - Domain Specific Language – framework
    - Test scripting
- Reasonably high-level (compared to C)
  - Good for software maintenance

Erlang in Practice – Pros

- Dynamic typing
  - Aids rapid prototyping
- OTP – includes useful building blocks
  - Supervisor
  - Generic server
  - Finite state machine
  - Unit test
  - Soon?: build and package system

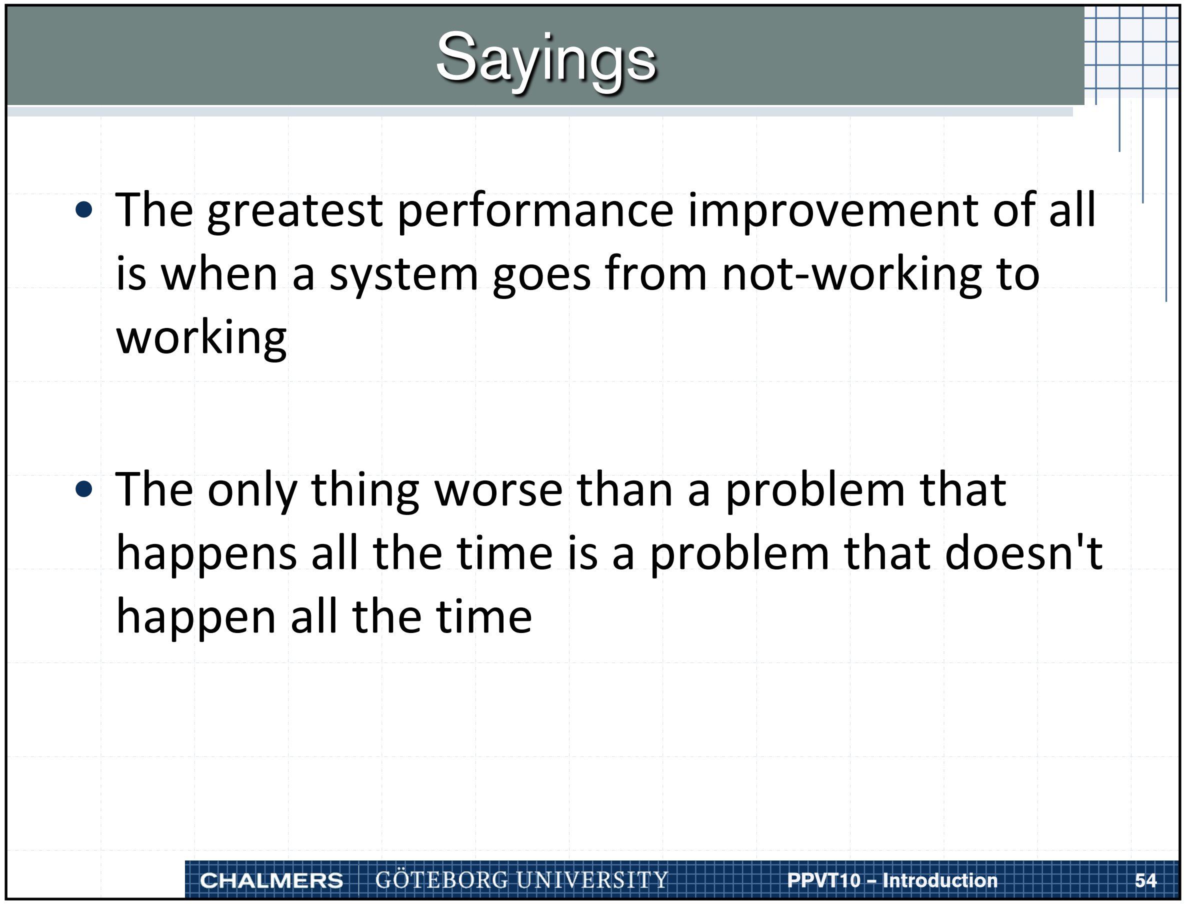
Erlang in Practice – Cons

- Hard to find good Erlang programmers (?)
  - Management b……t
  - Long live Chalmers/GU
- A bit too low-level language
  - Given current HW limitations one must sometimes optimise to the point where the code is not portable (with the same performance)
  - Raise the abstraction and provide a customisable compiler, VM (Elixir?)

Erlang in Practice – Cons

- Where is the type system?
  - A static type system of Haskell-nature would probably be a hindrance
  - But good static analysis tools are desperately needed
  - Types are an excellent form of documentation

More Than True



Sayings

- The greatest performance improvement of all is when a system goes from not-working to working
- The only thing worse than a problem that happens all the time is a problem that doesn't happen all the time

CHALMERS GÖTEBORG UNIVERSITY PPVT10 – Introduction 54

Functional Programming

- Widespread use
  - Embedded (cars, satellites, etc.), web-apps, games, banks, big-data, …
- Abstractions and compositionality
- Productivity gains

The Industrial Experience

- It is more difficult that you expect, but
  - Usually not in complexity but size
- Good methodical approach helps
- Lateral thinking is an asset
  - Learn many programming paradigms
  - Learn many programming languages