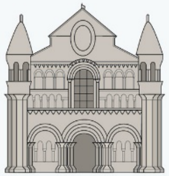
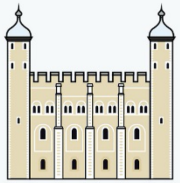
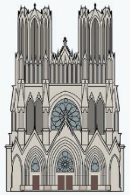
Architectural Styles – Part III

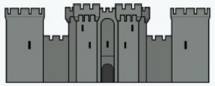
Truong Ho-Quang

truongh@chalmers.se


ROMANESQUE


NORMAN

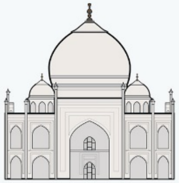

GOTHIC

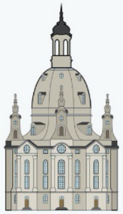

MEDIEVAL

RENAISSANCE

TUDOR

ELIZABETHAN


INDOISLAMIC

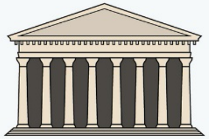

BAROQUE

JACOBEAN

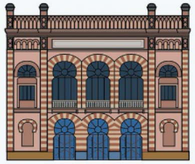
PALLADIAN

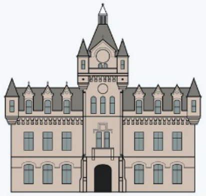
ROCOCO

GEORGIAN


NEOCLASSICAL

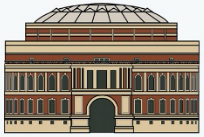
GOTHIC REVIVAL

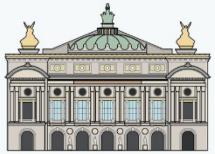

MOORISH REVIVAL

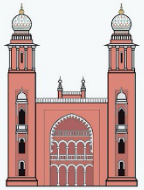

BARONIAL

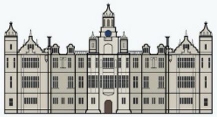
FEDERAL

REGENCY


ITALIANATE


EMPIRE


INDOSARACENIC


JACOBETHAN

CHICAGO SCHOOL

Schedule

Week		Date	Time	Lecture	Note
3	L1	Wed, 20 Jan	10:15 – 12:00	Introduction & Organization	Truong Ho
3	L2	Thu, 21 Jan	13:15 – 15:00	Architecting Process & Views	o
4		Tue, 26 Jan	10:15 – 12:00	**Skip**	
4	S1	Wed, 27 Jan	10:15 – 12:00	<< Supervision: Launch Assig	
4	L3	Thu, 28 Jan	13:15 - 15:00	Roles/Responsibilities & Functional Deco	o
5	L4	Mon, 1 Feb	13:15 – 15:00	Architectural Styles P1	o
5	S2	Wed, 3 Jan	10:15 – 12:00	<< Supervisio /Assignment>>	TAs
5	L5	Thu, 4 Jan	13:15 – 15:00	Architectural Styles P2	Sam Jobara
6	L6	Mon, 8 Feb	13:15 – 15:00	Architectural Styles P3	Truong Ho
6	S3	Wed, 10 Feb	10:15 – 12:00	<< Supervision/Assignment>>	TAs
6	L7	Thu, 11 Feb	13:15 – 15:00	Design Principles (Maintainability, Modifiability)	Truong Ho
7	L8	Mon, 15 Feb	13:15 – 15:00	Performance – Analysis & Tactics	Truong Ho
7	S4	Wed, 17 Feb	10:15 – 12:00	<< Supervision/Assignment>>	TAs
7	L9	Thu, 18 Feb	13:15 – 15:00	Tactics: Reliability, Availability, Fault Tolerance	TBD
8	L10	Mon, 22 Feb	13:15 – 15:00	Guest Lecture 1	TBD
8	S5	Wed, 24 Feb	10:15 – 12:00	<< Supervision/Assignment>>	TAs
8	L11	Thu, 25 Feb	13:15 – 15:00	Guest Lecture 2	TBD
9	L12	Mon, 1 Mar	13:15 – 15:00	Reverse Engineering & Correspondence	Truong Ho
9	S6	Wed, 3 Mar	10:15 – 12:00	<< Supervision/Assignment>>	TAs
9	L13	Thu, 4 Mar	13:15 – 15:00	To be determined (exam practice?)	Truong Ho
9		Fri, 5 Mar	Whole day	Group presentation of Assignment (TBD)	Teachers
11	Exam				

We are HERE!

Assignment schedule

Week		Date	Lecture	Assignment 1 – Task 1 (A1T1)	Assignment 1 – Task 2 (A1T2)	Assignment 2 (A2)
3	L1	Wed, 20 Jan	10:15 – 12:00			
3	L2	Thu, 21 Jan	13:15 – 15:00			
4		Tue, 26 Jan	10:15 – 12:00			
4	S1	Wed, 27 Jan	10:15 – 12:00	Launch A1T1		
4	L3	Thu, 28 Jan	13:15 - 15:00			
5	L4	Mon, 1 Feb	13:15 – 15:00			
5	S2	Wed, 3 Jan	10:15 – 12:00	Work A1T1		
5	L5	Thu, 4 Jan	13:15 – 15:00			
6	L6	Mon, 8 Feb	13:15 – 15:00			
6	S3	Wed, 10 Feb	10:15 – 12:00	Work A1T1		
6	L7	Thu, 11 Feb	13:15 – 15:00	Hand-in A1T1 Peer Rev A1T1	A1T2 released	
7	L8	Mon, 15 Feb	13:15 – 15:00			
7	S4	Wed, 17 Feb	10:15 – 12:00	Hand-in PR A1T1	MQTT intro	A2 released
7	L9	Thu, 18 Feb	13:15 – 15:00			
8	L10	Mon, 22 Feb	13:15 – 15:00			
8	S5	Wed, 24 Feb	10:15 – 12:00		Work A1T2	
8	L11	Thu, 25 Feb	13:15 – 15:00			
9	L12	Mon, 1 Mar	13:15 – 15:00			
9	S6	Wed, 3 Mar	10:15 – 12:00		Work A1T2	Hand-in A2
9	L13	Thu, 4 Mar	13:15 – 15:00			
9		Fri, 5 Mar	Whole day		Present A1T2	
11	Exam					

Hand-in
deadline
In 3 days

Clarification – A1T1

- Question 4: “following functionalities”
  - is a typos
  - I meant: “following the steps described in a), b), c) in order to reason about their design and component diagram(s)”
- “interesting animals”

  = animals of interest (rare animals, animals in danger, animals that are under population control…)
  - There is a list of these interesting animals.

Outline of Topics for Today’s Lecture

- Architectural Styles
  - Publish-Subscribe Style
  - Blackboard Style
  - Layered Style

CONTENTS

1. Introduction
2. Architectural styles
   - 2.1 Client/Server
   - 2.2 Pipe and Filter style

Lecture 4

   - 2.3 Blackboard style
   - 2.4 Publish Subscribe
   - 2.5 Layered style

TODAY!

   - 2.6 Peer-to-Peer style
   - 2.7 Microservices style
   - 2.8 Event-Driven style
3. Conclusions

Lecture 5

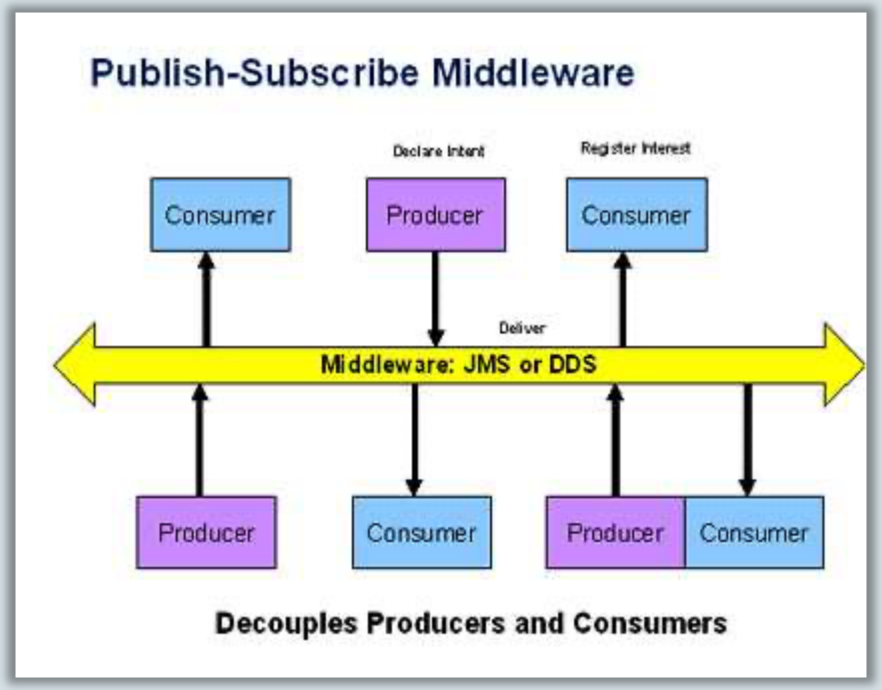
CONTENTS

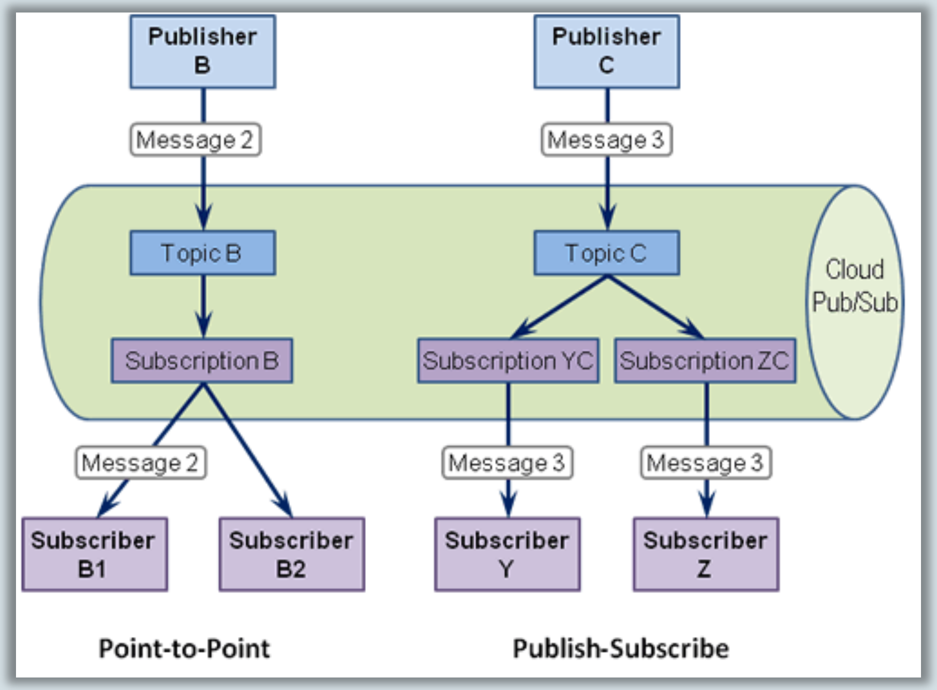
1. **Introduction**

2. Architectural styles
    2.1 Client/Server
    2.2 Pipe and Filter style
    2.3 Blackboard style
    2.4 Publish Subscribe
    2.5 Layered style
    2.6 Peer-to-Peer style
    2.7 Microservices style
    2.8 Event-Driven style

3. Conclusions

Publish–Subscribe

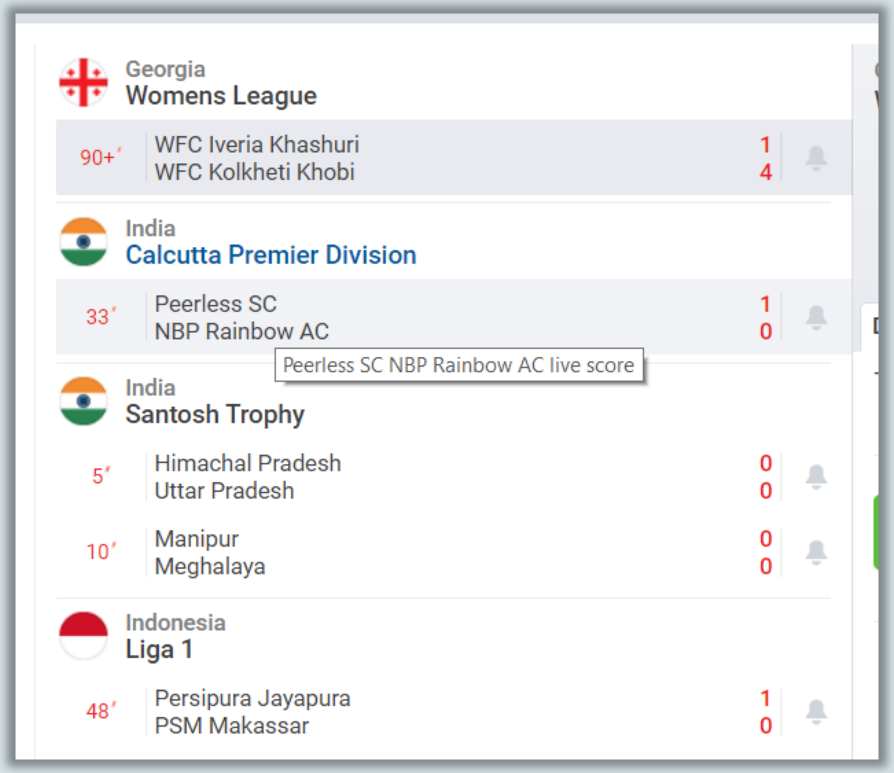




Publish-Subscribe

P/S is like: subscriptions that you know:

e.g. newspapers or live sports highlights:



Georgia
Womens League

90+'	WFC Iveria Khashuri WFC Kolkheti Khobi	1 4

India
Calcutta Premier Division

33'	Peerless SC NBP Rainbow AC	1 0

Peerless SC NBP Rainbow AC live score

India
Santosh Trophy

5'	Himachal Pradesh Uttar Pradesh	0 0
10'	Manipur Meghalaya	0 0

Indonesia
Liga 1

48'	Persipura Jayapura PSM Makassar	1 0

Publish-Subscribe

- Components interact via announced messages, or events.
  - Components may subscribe to a set of events.
  - It is the job of the publish-subscribe runtime infrastructure to make sure that each published event is delivered to all subscribers of that event.

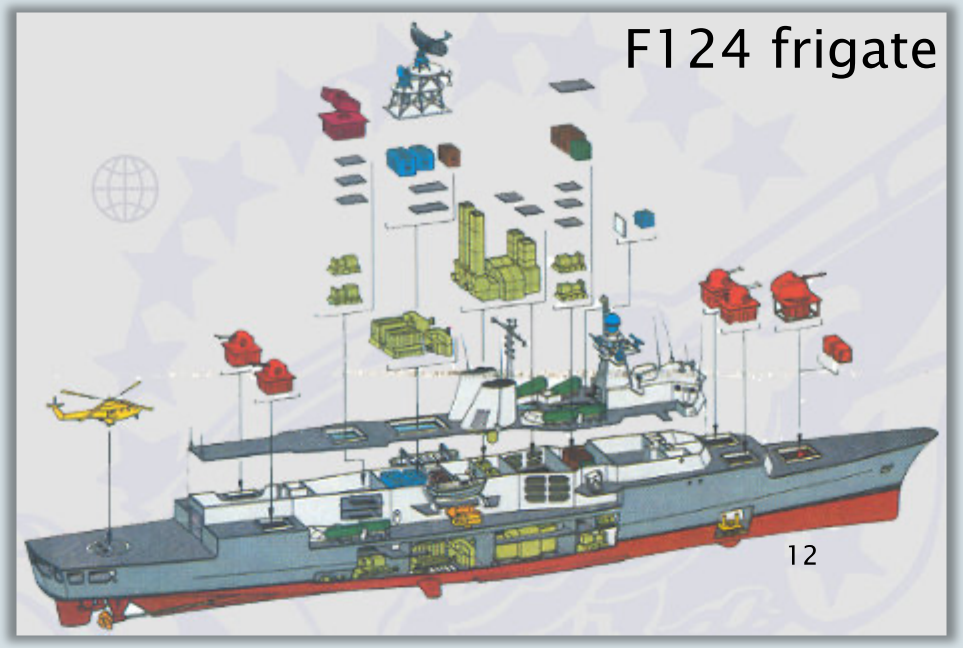
- **Advantages**: loose coupling, scalability, extendibility, improved security (messages sent to subscribers only)
- **Limitations**: need to guarantee delivery, performance problems when overloaded with messages

Publish-Subscribe Style
Case Study: SPLICE

Developed by Thales (formerly Hollandse Signaal App.)

Oriented towards high quality control systems:

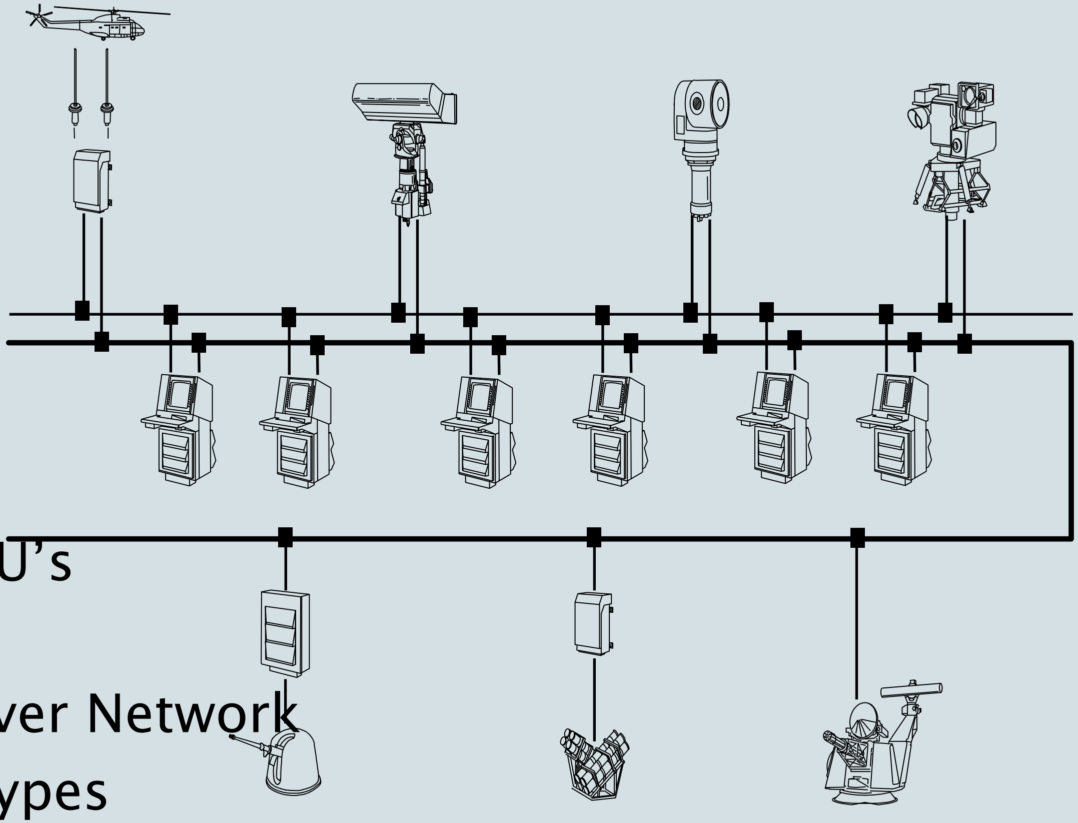
- Distributed
- Fault tolerant (support of degraded modes)
- (Soft) real-time
- Extensible


F124 frigate

Architecture Requirements

The architecture is characterized as:

- real-time
- distributed
- data driven
- fault tolerant

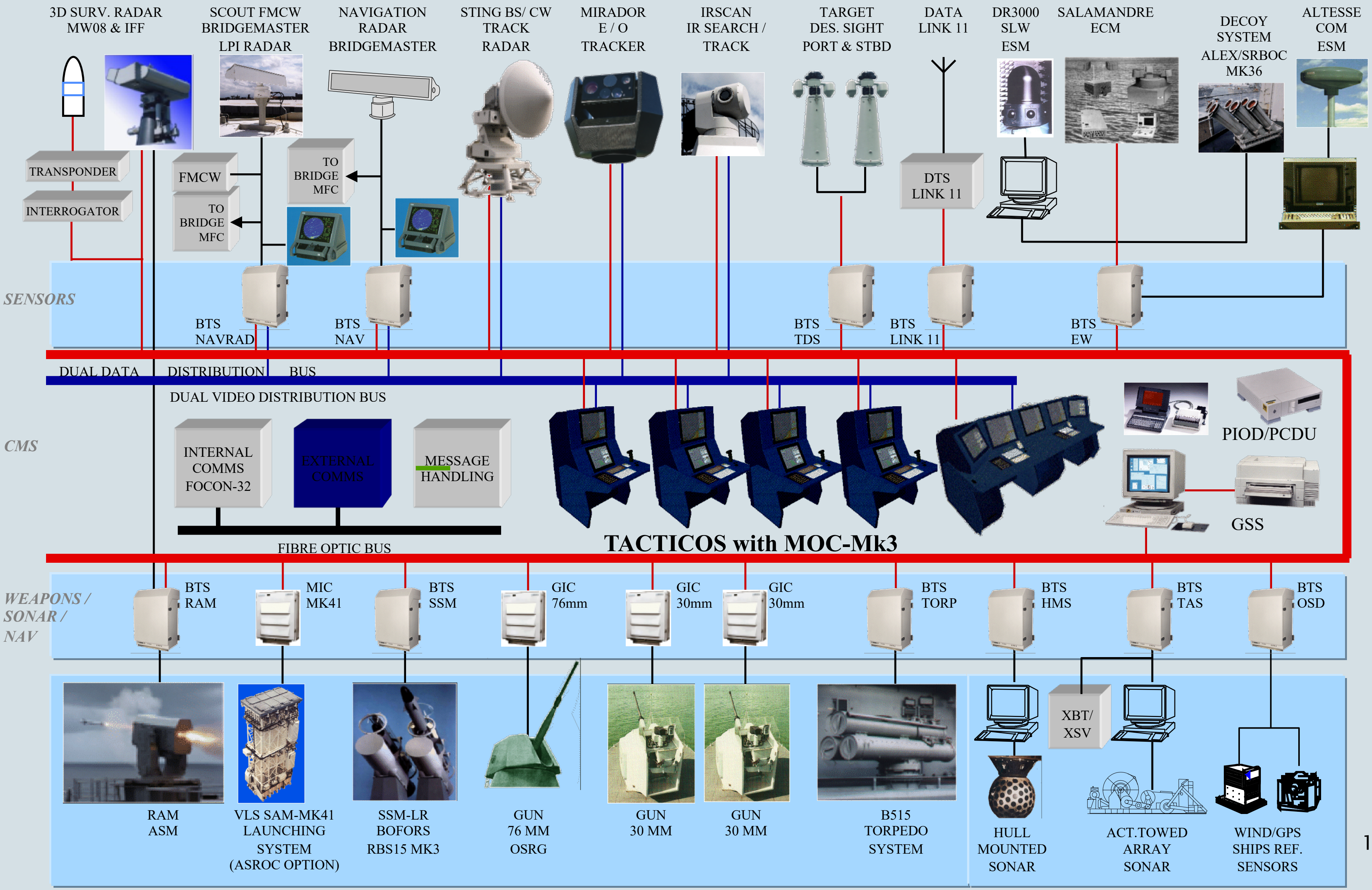


with some typical figures:

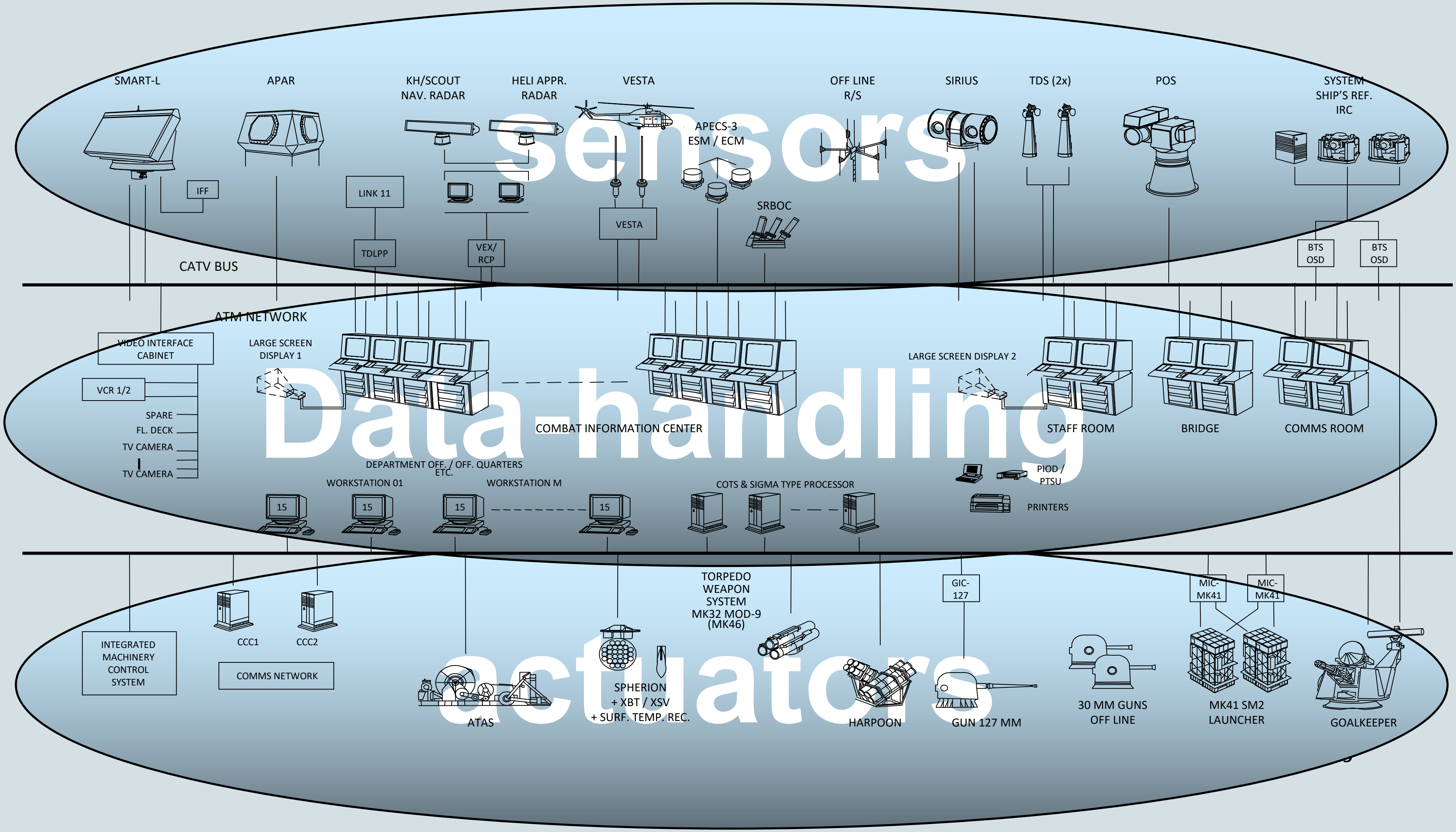
50 nodes containing 170 CPU’s

- 2200 active executables
- 4000 Hz. data-updates over Network
- >2000 distributed data-types

Configuration example: frigate size system



Combat-Management-System Overview



SPLICE Application Domain

Used in command and control & traffic mgm. systems

Typical process:

1. **Acquire input**–signals through sensors
2. **Process input**–signals
3. **Interpret input** in terms of environment model
4. **Take action** through effectors
   or support operators in decision making

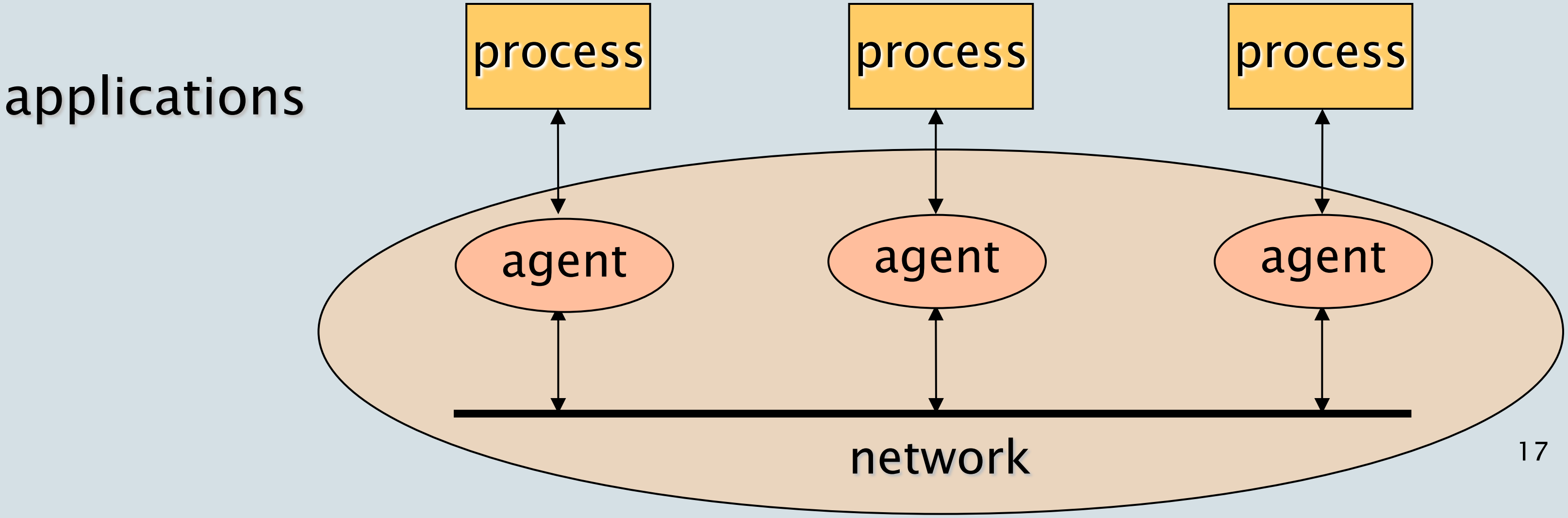
The interpretation of input may require the **sharing of data between many different applications** that **act in irregular patterns**

SPLICE Pub/Sub-Model

Applications are concurrently executing processes that implement part of the overall functionality

Processes register with network agents whether they are producers or consumers of a type of data.

The network agents manage distribution of data.



SPLICE Data Sorts

- Data elements are labeled records.
- Each record has a system-wide unique label, called the *data sort*
- A field of a sort may be declared ***key*** if it uniquely determines the values of the non-key fields

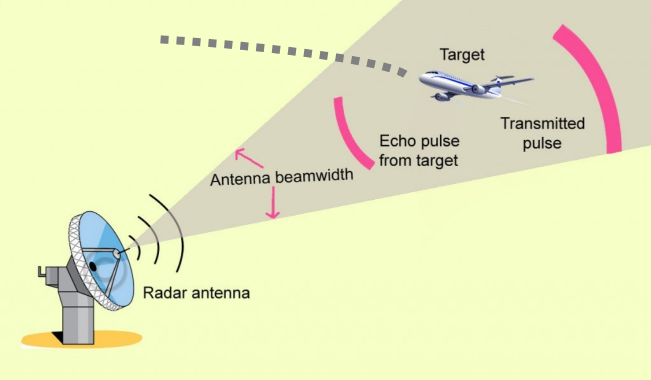
```
sort flightplan
    key flightnumber : string
    Departure        : time
    Arrival          : time
    Aircraft         : string
```

```
sort track
    key flightnumber : string
    key index        : integer
    State            : string
```

SPLICE Example (1/5)

Consider a system for tracking flying objects:

- Observations are made by a radar (and are called plots), i.e. the acquisition sensor
- Plots are correlated into tracks, that are interpreted in terms of a flight trajectory model
- Tracks are used to control the direction of the radar and for taking action through effectors)



SPLICE Example (2/5)

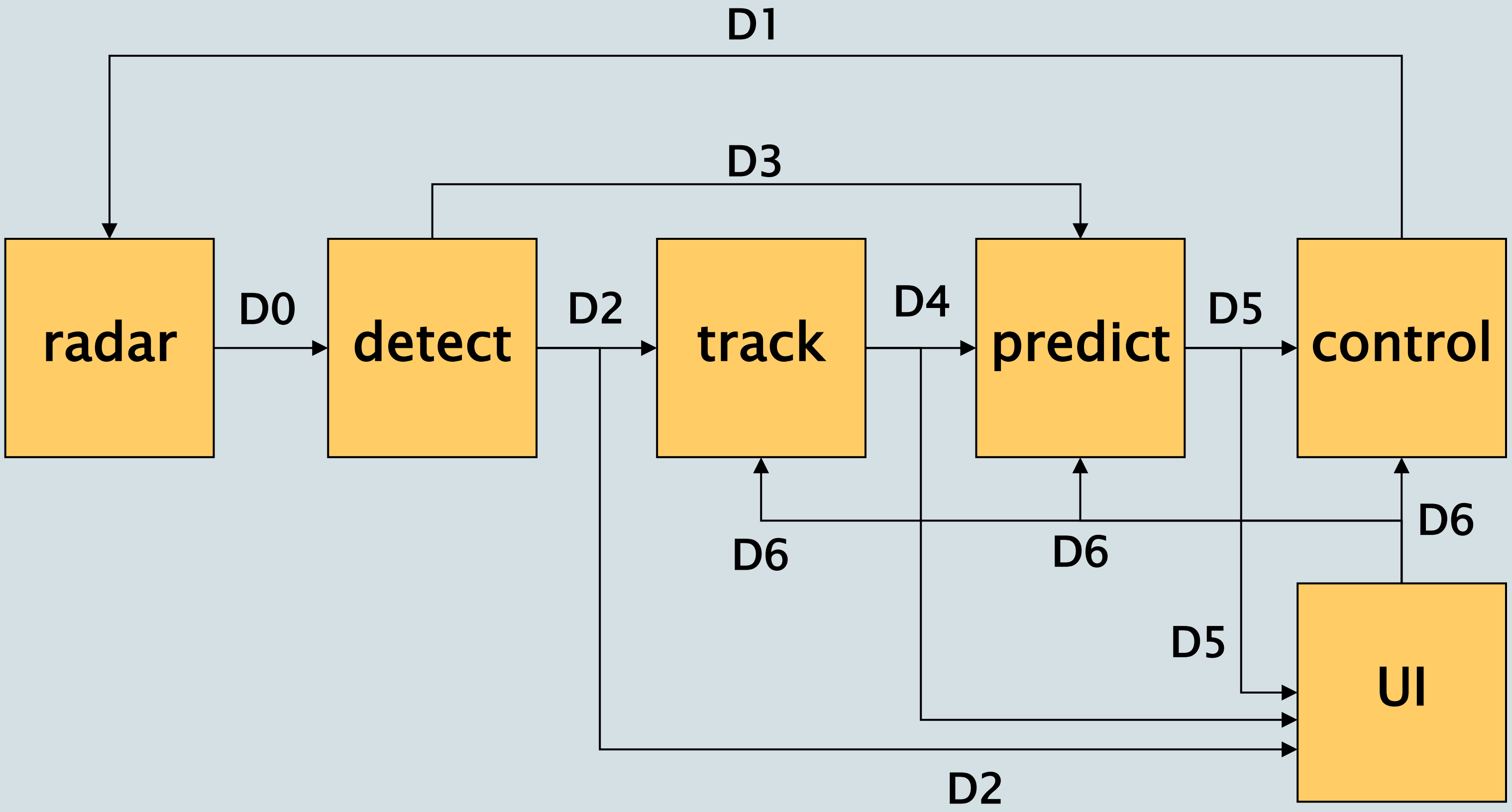
Processes

Radar: generates signals
Detect: processes radar signals into plots
Track: correlates plots into tracks
Predict: predicts next coordinates of the flying object
Control: the radar to probe the next position of the object
UI: user interface of the system

Sorts (=data types)

D0: radar signals
D1: control data to the radar
D2: plots (coordinates) from the radar
D3: sensor characteristics
D4: speed-vector of the object
D5: predicted object coordinates
D6: user commands

Application model using one-to-one connections



SPLICE Example program (4/5)

```
Program Detect
  sort raw_data: radar_signals consumed
  sort obj_pos: coordinates produced
  signal: sensor_data

Forever do
   signal := get(raw_data);
   if valid_signal(signal)
       then obj_pos:= f(signal); put(obj_pos)
       else  { corrective action }
End program Detect
```

What happens when multiple copies of *Detect* are running concurrently?

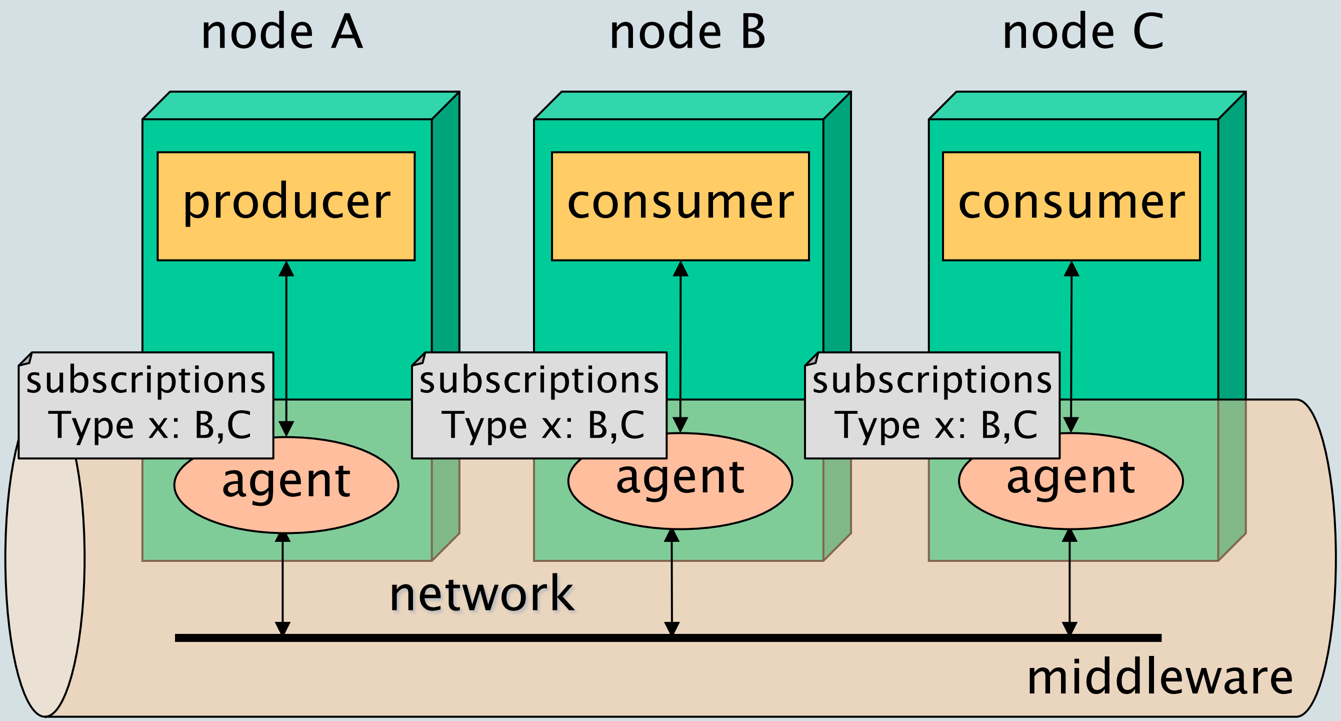
SPLICE Example (5/5)

```
Program Predict
  sort radar_attr: sensor_attr consumed
  sort track_data: track consumed
  sort pred_coord: coordinates produced
  sort user_cmnd: command consumed
  result: integer
  local_track: track

get(radar_attr);
Forever do
   result := get(track_data);
   if valid_track(result)
      then
        { local_track:=predict new coordinates};
        put(obj_pos)
      else
        if local_track.timestamp + radar_attr.cycle_time >
           time – comm_delay
        then { new data too late; corrective action };
   result := get(user_cmnd);
   if valid_cmnd(result)
       then  { deal with command }

End program Predict
```

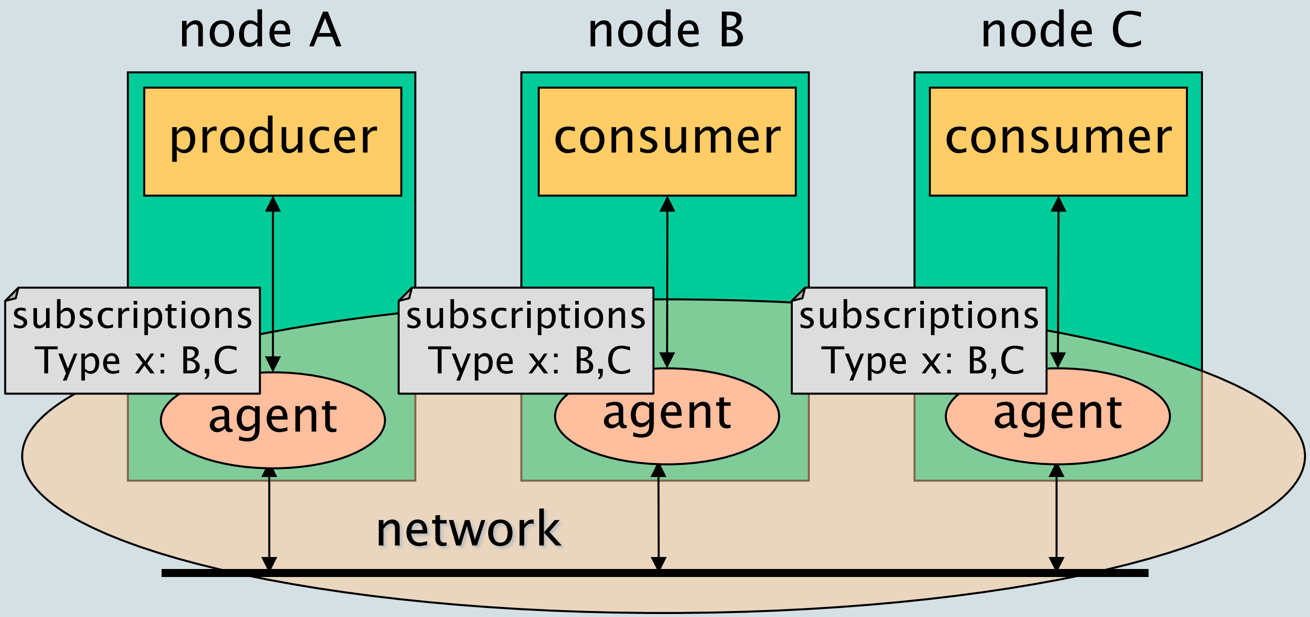
P/S Deployment



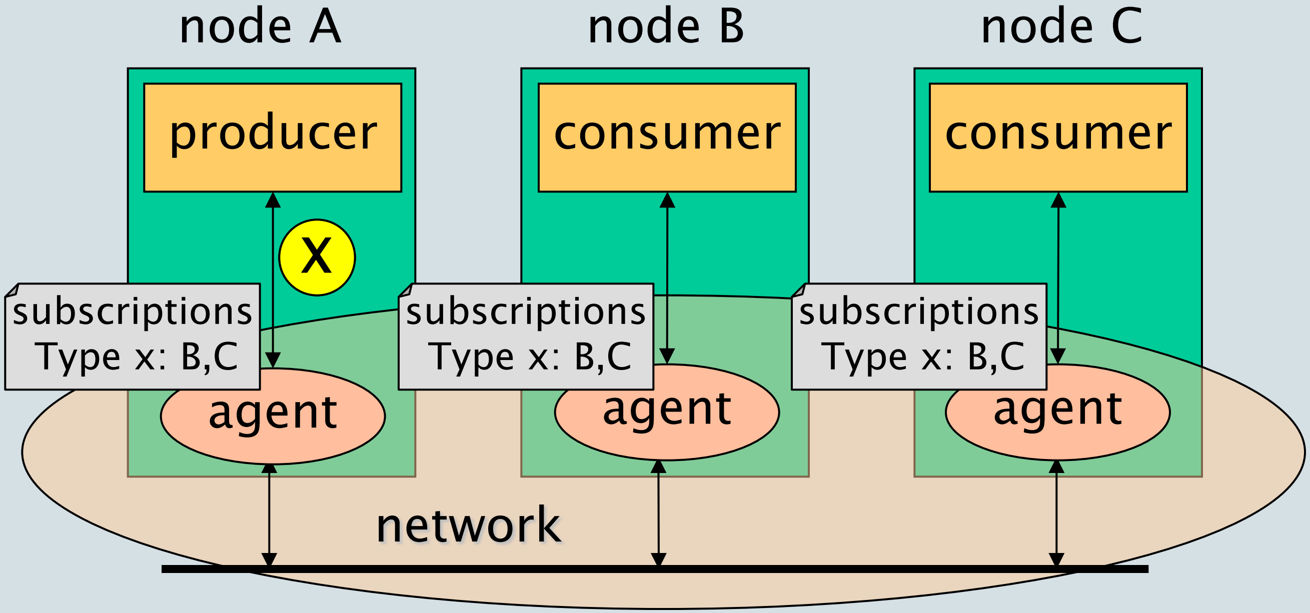
producer = application software component

agent = middleware software component

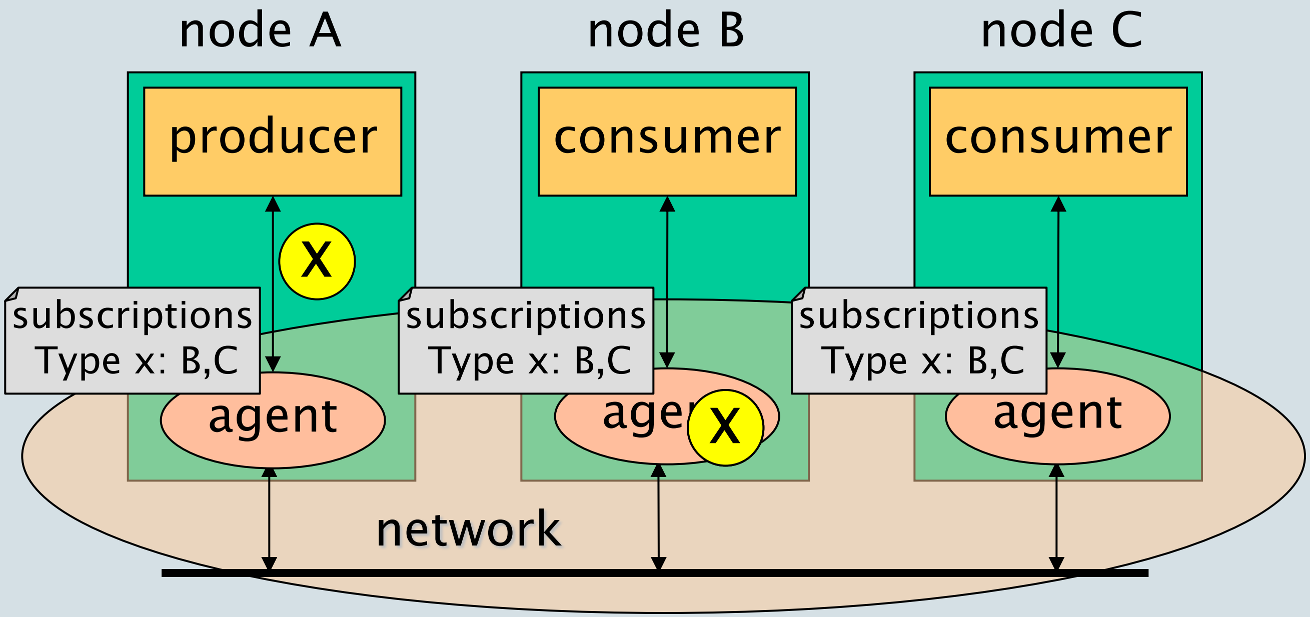
Registration Phase



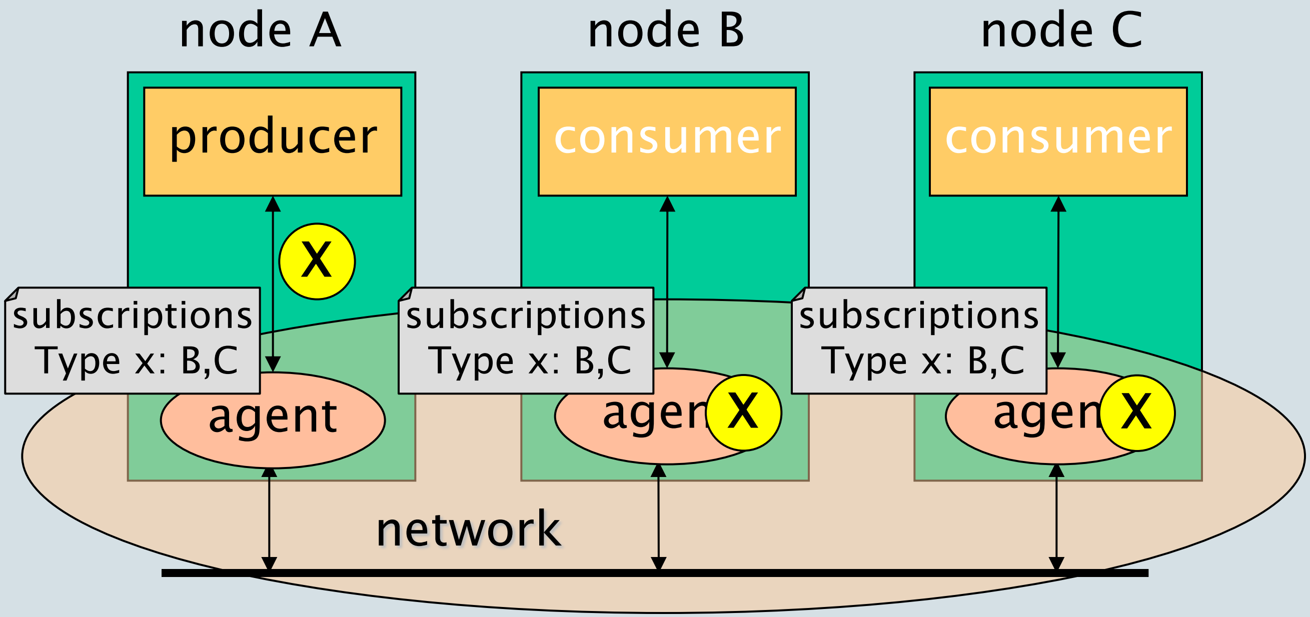
Distribution Phase



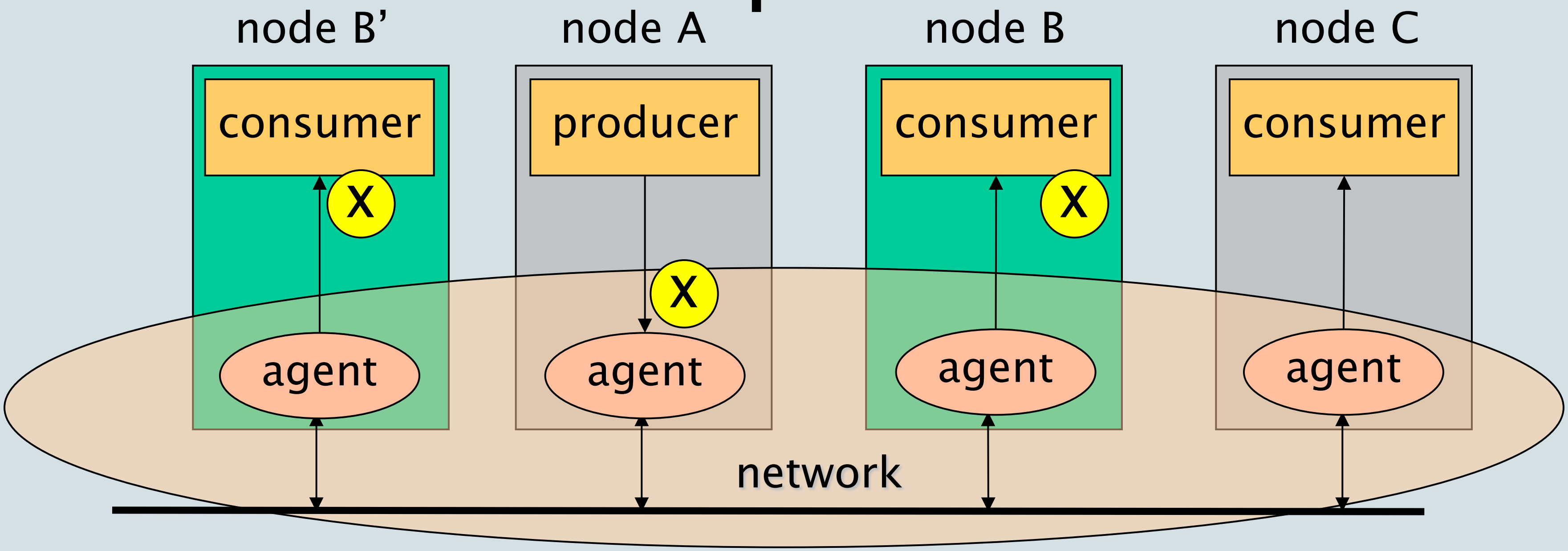
Distribution Phase



Distribution Phase



Insert New Consumer Component

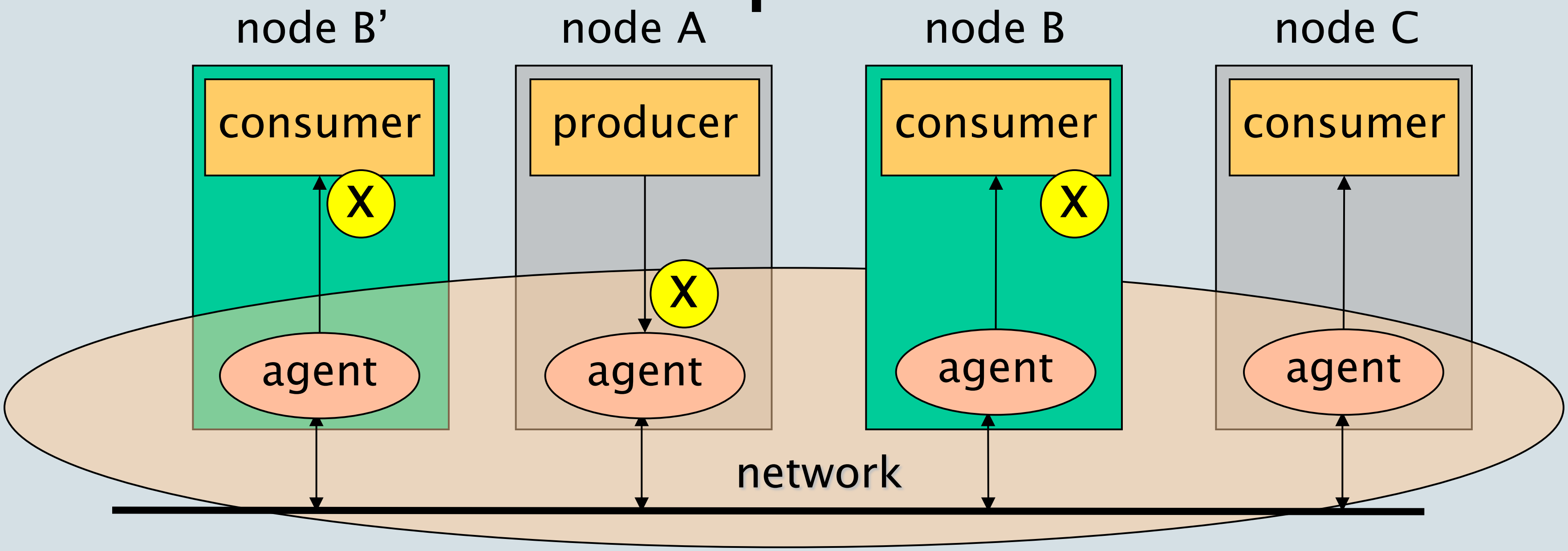


B’ can build up state from inputs it receives.

If B and B’ both consume and produce data, then duplicate data is generated.

B’ can monitor output of B to check convergence

Phase out old Consumer Component

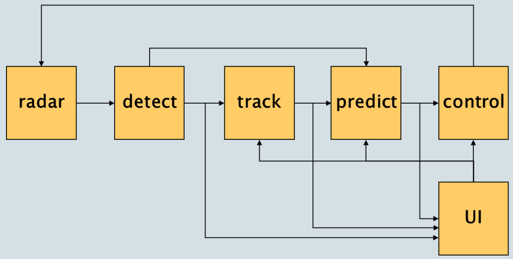


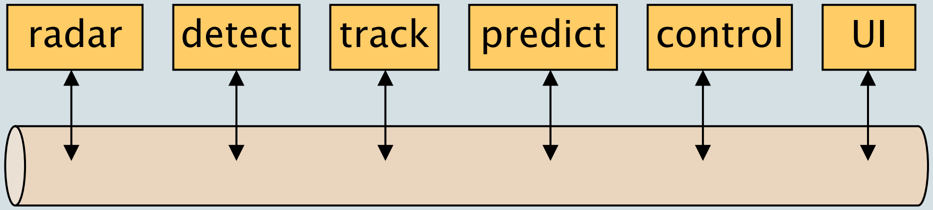
Once B’ has converged with B, B is stopped

Reflection on Architectural Style of Pub/Sub

The architectural style strongly influences

- the complexity of the overall design, and
- the systems' quality attributes





When to use P/S

- Data is short-lived
- ‘Frequent’ production of data
- Consumers are interested in updates
- Multiple consumers
- Dynamically changing topology of producers and/or consumers

References

Control System Software, M. Boasson
IEEE Transactions on Automatic Control, Vo. 38, No. 7, July 1993

Software Architecture for Large Embedded Systems
M. Boasson and E. de Jong
http://www.cwi.nl/~marcello/SAPapers/BJ97.html

CONTENTS

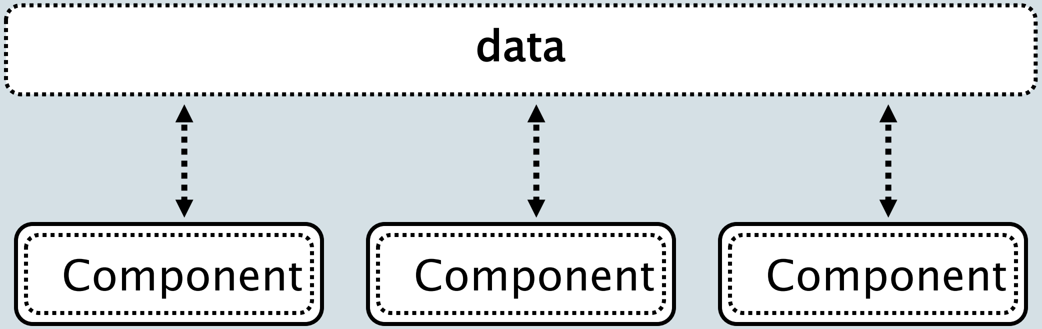
1. **Introduction**

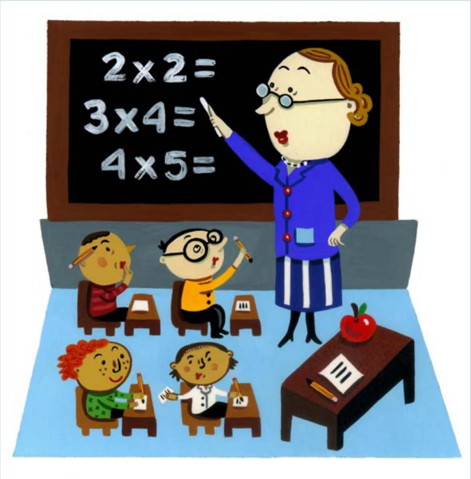
2. Architectural styles
    - 2.1 Client/Server
    - 2.2 Pipe and Filter style
    - 2.3 Blackboard style
    - 2.4 Publish Subscribe
    - 2.5 Layered style
    - 2.6 Peer-to-Peer style
    - 2.7 Microservices style
    - 2.8 Event-Driven style

3. Conclusions

Blackboard Style (1)

Concept: Concurrent transformations on shared data





Components: processing units (typically knowledge source)

Connectors: blackboard
interaction style: asynchronous

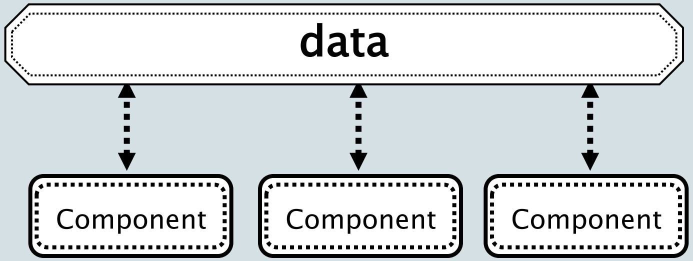
Topology: one or more transformation-components may be connected to a data-space,
there are typically no connections between processing units (bus-topology)

Blackboard Style (2)

Behaviour Types:

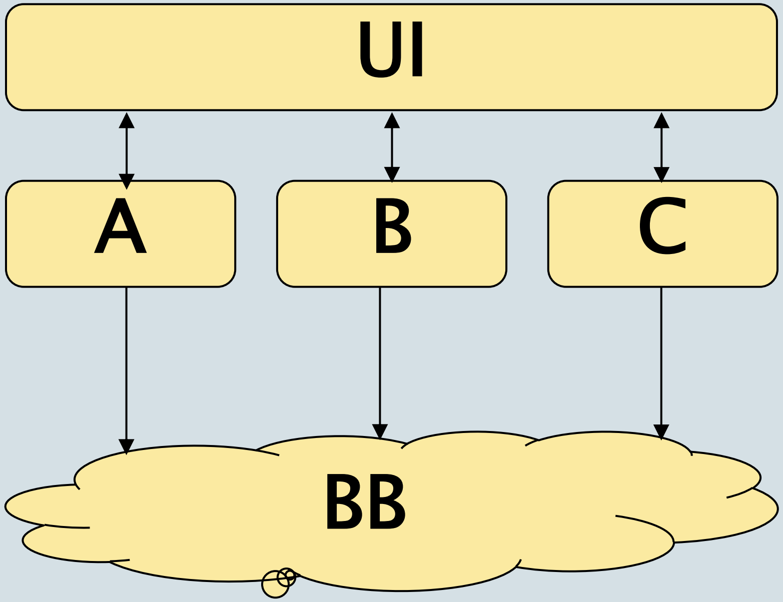
a. **Passive repository**
Accessed by a set of components; e.g. database or server

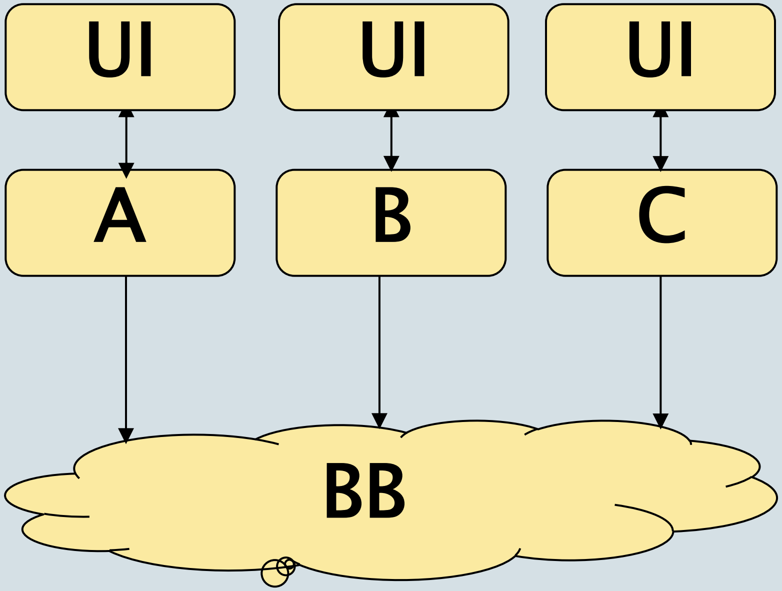
b. **Active repository**
Sends notification to components when data of interest changes; e.g. blackboard or active database



Constraints:
Consistency of repository: Various types of (transaction) consistency

Layering & Blackboard





Blackboard Style (3)

Advantages:

- Allows different control heuristics
- Reusable & heterogeneous knowledge sources
- Support for fault tolerance and robustness by adding redundant components

+/- Dataflow is not directly visible

Disadvantages

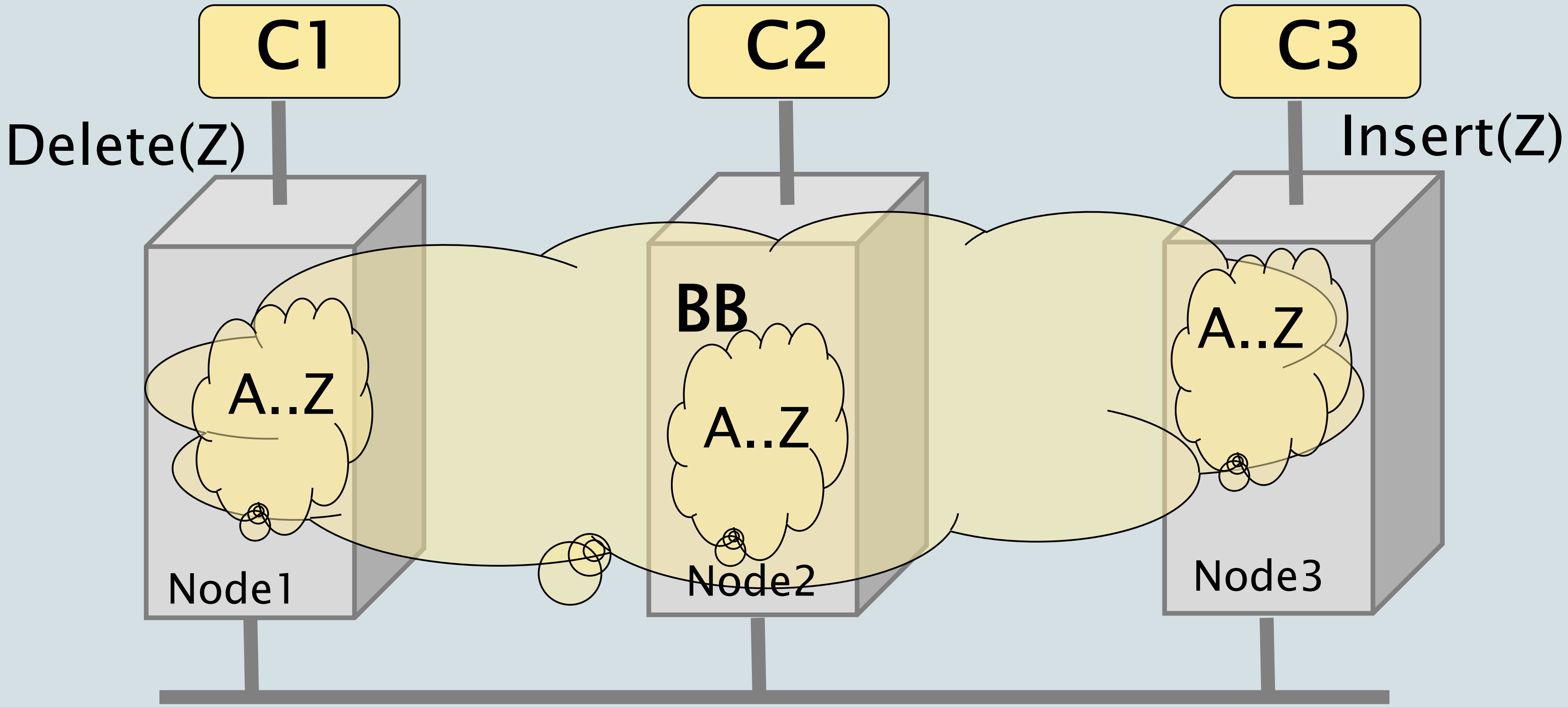
- Distributed implementation is complex
  - distribution and consistency issues

Blackboard Characteristics

- Data may be structured (DB) or unstructured
- Data may be selected based on content
- Applications may insert/retrieve different data-type per access.

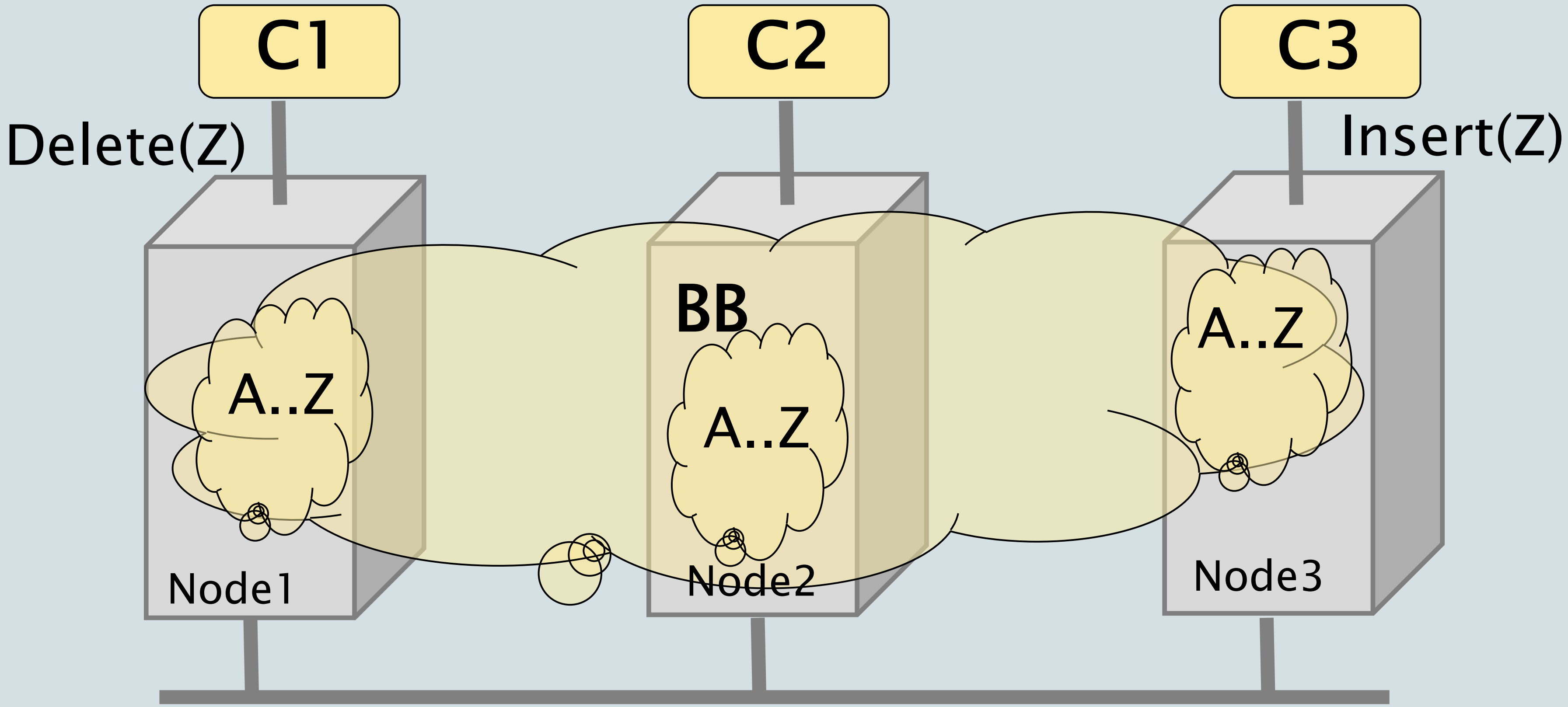
  This in contrast to pub-sub where data of the same type is retrieved repeatedly

Blackboard and consistency



Node 1, 2 and 3 are all storing a copy of the entire dataset (A–Z). This increases reliability & availability and improves response time *(. But ….

Blackboard and consistency



C1 and C3 may ‘see’ a different content on the blackboard depending on the order (and speed) of executing the delete and insert actions.

Example of Blackboard Architecture

- Hearsay, speech understanding

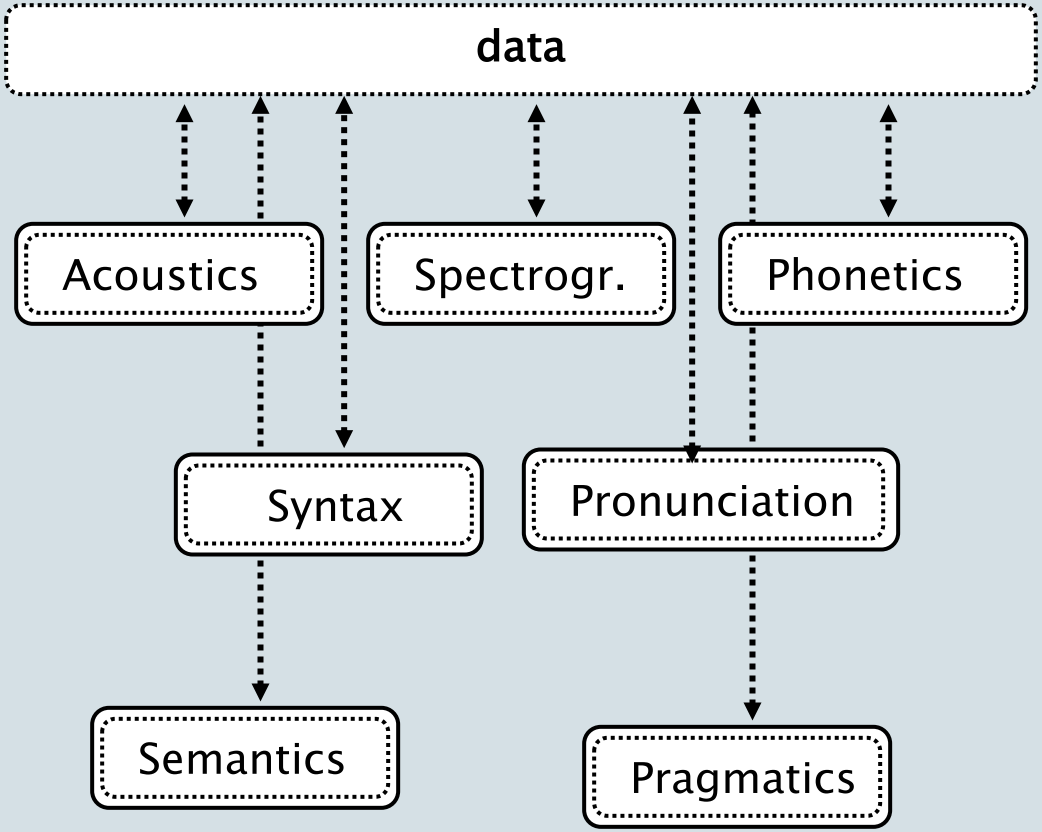
- Hearsay was developed in the 1970’s by Raj Reddy et al. at Carnegie Mellon University.

- Randy Davis, *Speech Understanding Using Hearsay,* MIT videotape, 1984.

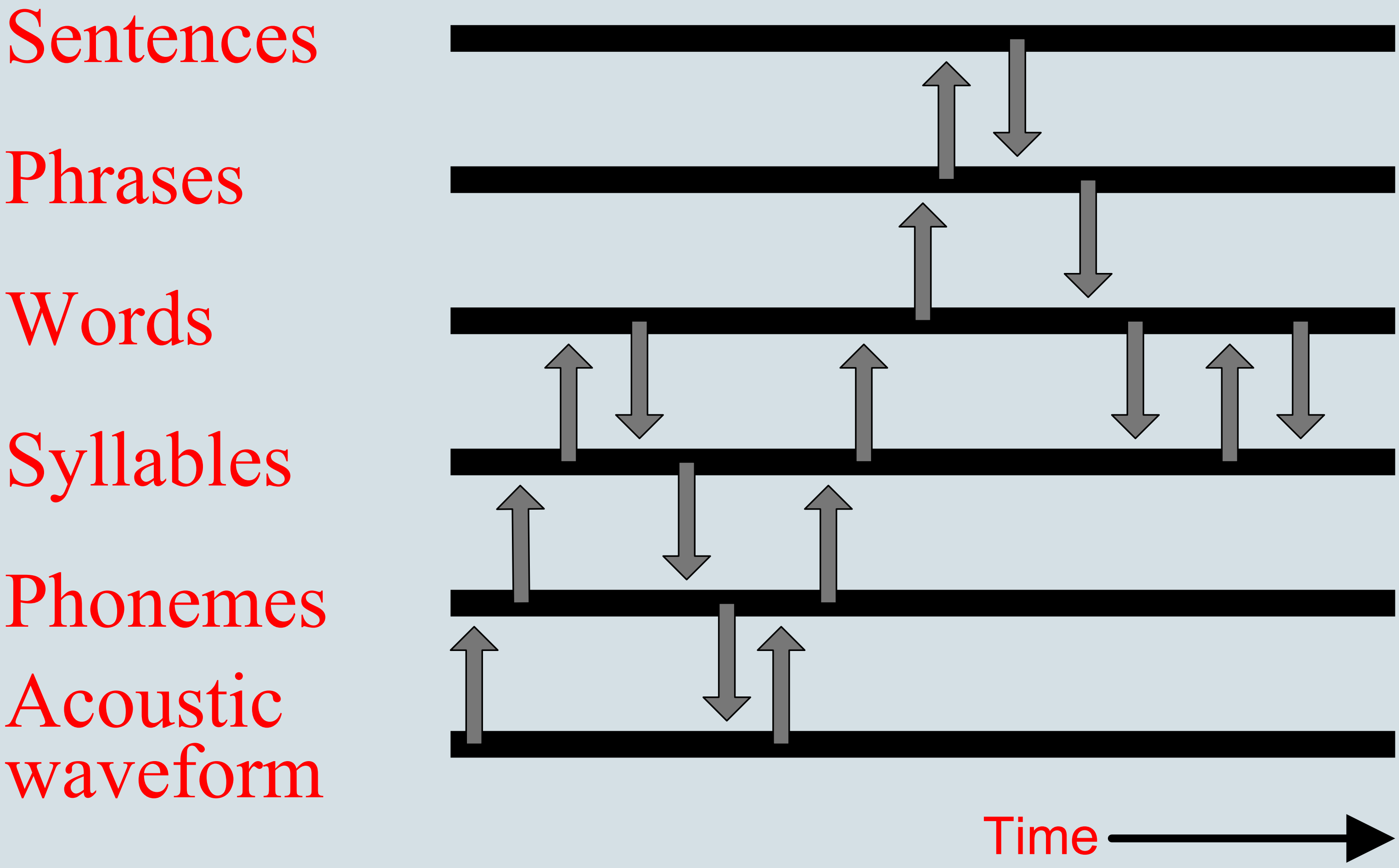
Slides adapted from Terry Bahill, Univ. Arizona, 2007

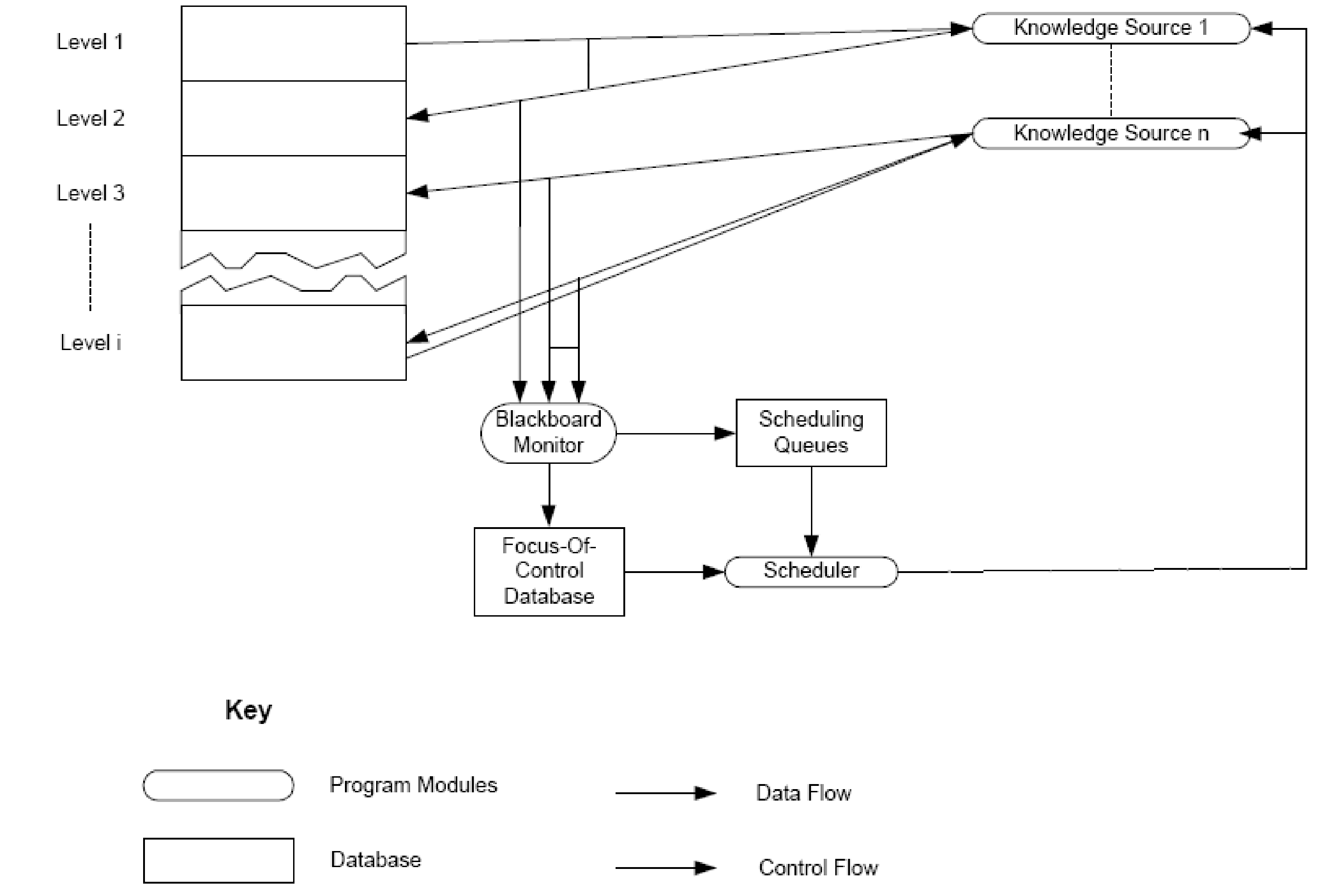
Hearsay: knowledge sources

- Acoustics
- Spectrographs
- Phonetics
- Pronunciation
- Coarticulation
- Syntax
- Semantics
- Pragmatics



Hearsay: levels of abstraction*





- L.D. Erman, F. Hayes-Roth, V.R. Lesser and D. R. Reddy, “The Hearsay-II speech understanding system: integrating knowledge to resolve uncertainty”, ACM Computing Surveys 12(2), pp213-253, 1980.
- L.D. Erman, P.E. London and S. F. Fickas, “The Design and an Example Use of Hearsay-II”, Proc. IJCAI-81, pp 409-415, 1981.

Hearsay: control

- Data driven
- Asynchronous
- Opportunistic
- Islands of reliability
- Combined top–down and bottom–up

Blackboard Style (4) Quality Factors

Extensibility: components can be easily added

Flexibility: functionality of components can be easily changed

Robustness: + components can be replicated,
- blackboard is single point of failure

Security: - all process share the same data
+ security measures can be centralized around blackboard

Performance: easy to execute in parallel fashion
consistency may incur synchroniz.-penalty

Blackboard Style (5) Application Context

Rules of thumb for choosing blackboard (o.a. from Shaw):

- if representation & management of data is a central issue
- if data is long-lived
- if order of computation
  - can not be determined a-priori
  - is highly irregular
  - changes dynamically
- if units of different functionality (typically containing highly specialized knowledge) concurrently act on shared data (horizontal composition of functionality)

Example application domain: expert systems

CONTENTS

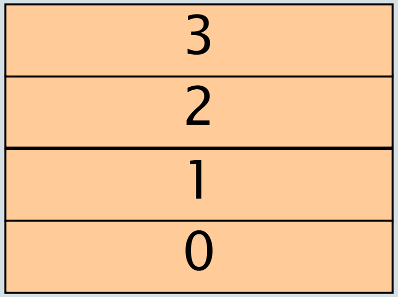
1. **Introduction**
2. Architectural styles
   - 2.1 Client/Server
   - 2.2 Pipe and Filter style
   - 2.3 Blackboard style
   - 2.4 Publish Subscribe
   - 2.5 Layered style
   - 2.6 Peer-to-Peer style
   - 2.7 Microservices style
   - 2.8 Event-Driven style
3. Conclusions

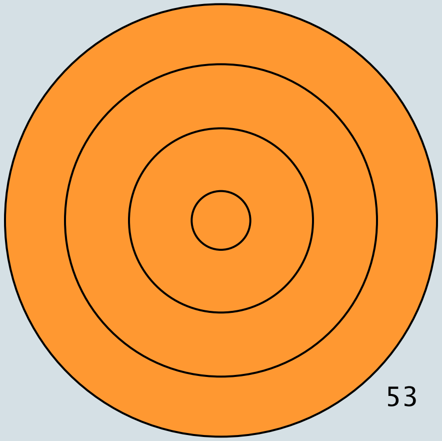
Layering (1)

Goals: Separation of Concerns, Abstraction, Modularity, Portability

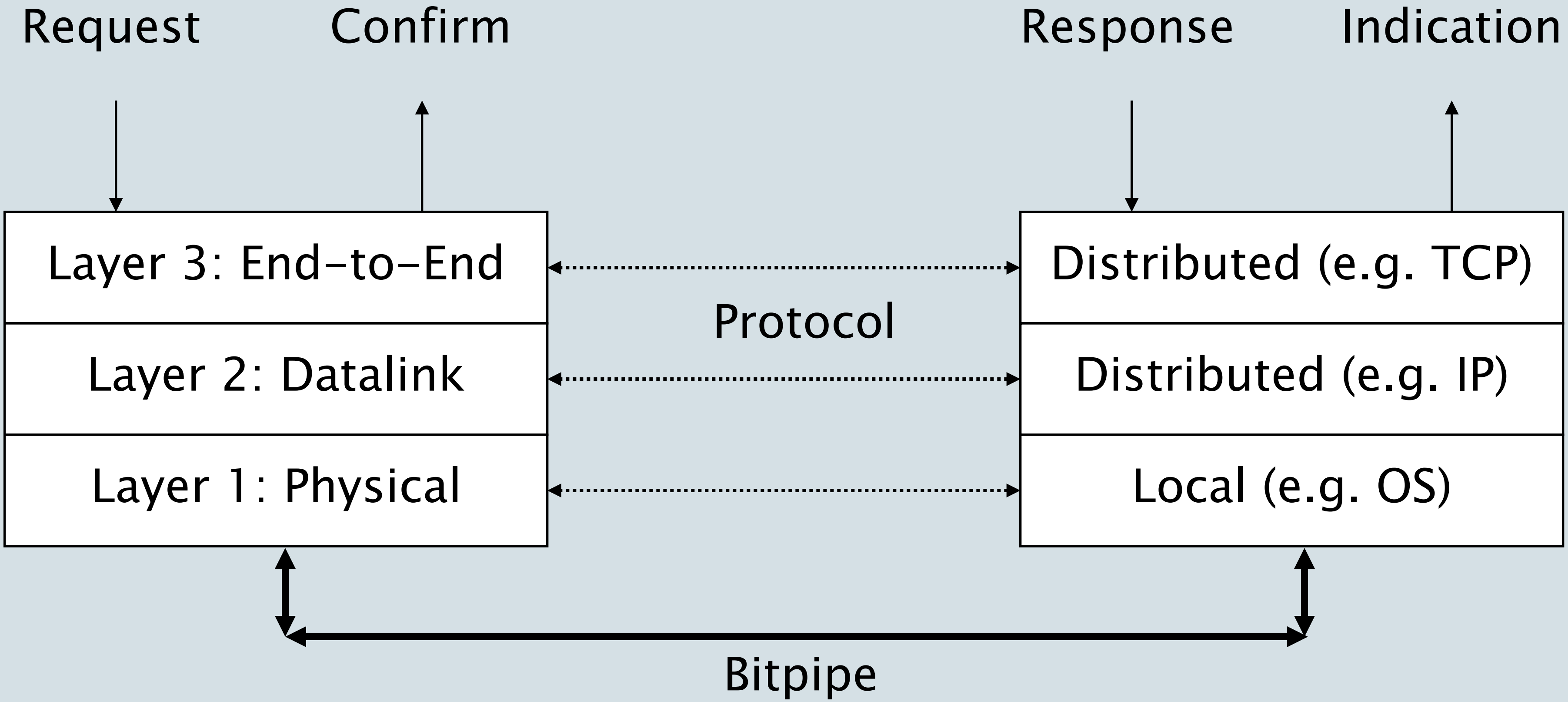
Partitioning in non-overlapping units that

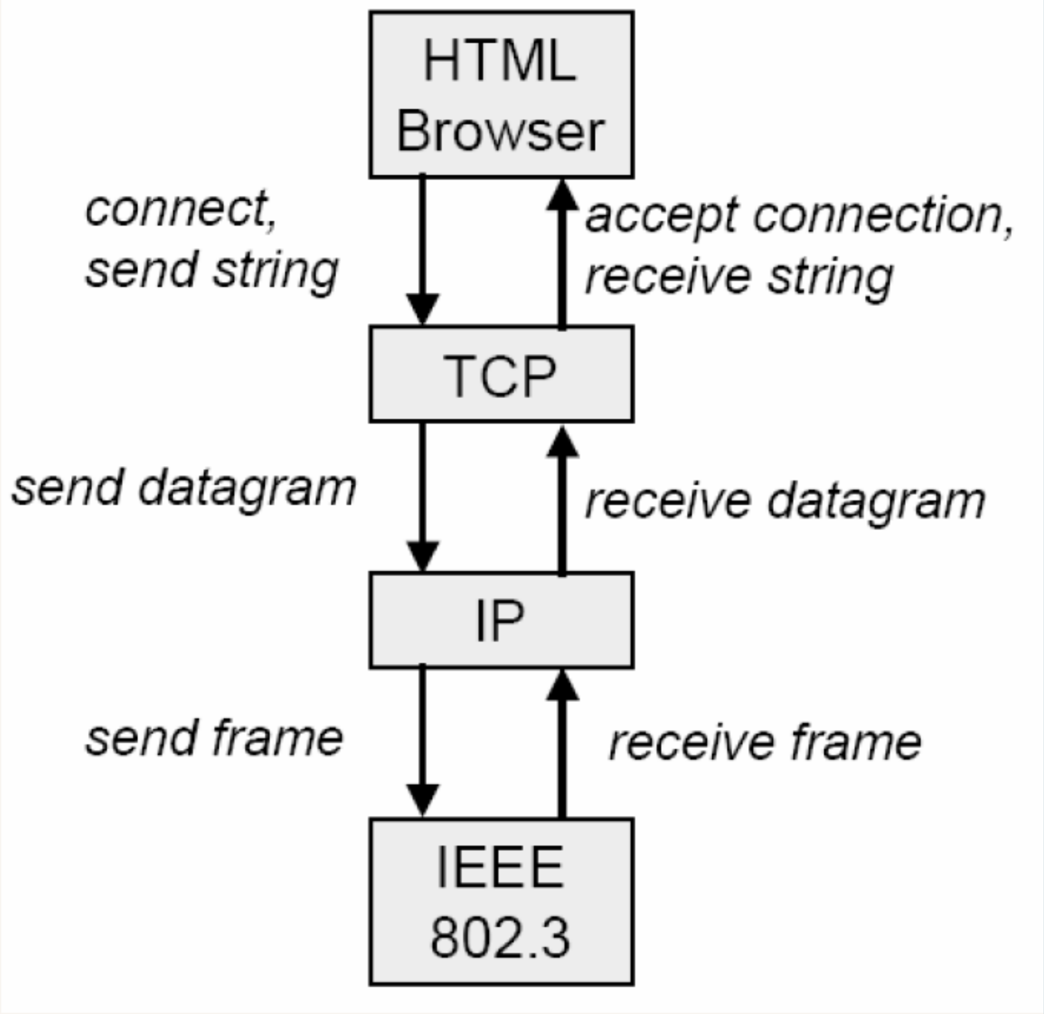
- provide a cohesive set of services at an abstraction level
  (while abstracting from their implementation)
- layer *n* is allowed to use services of layer *n-1*
  (and not vice versa)
  alternative:
    bridging layers: layer *n* may use layers <*n*
    enhances efficiency but hampers portability





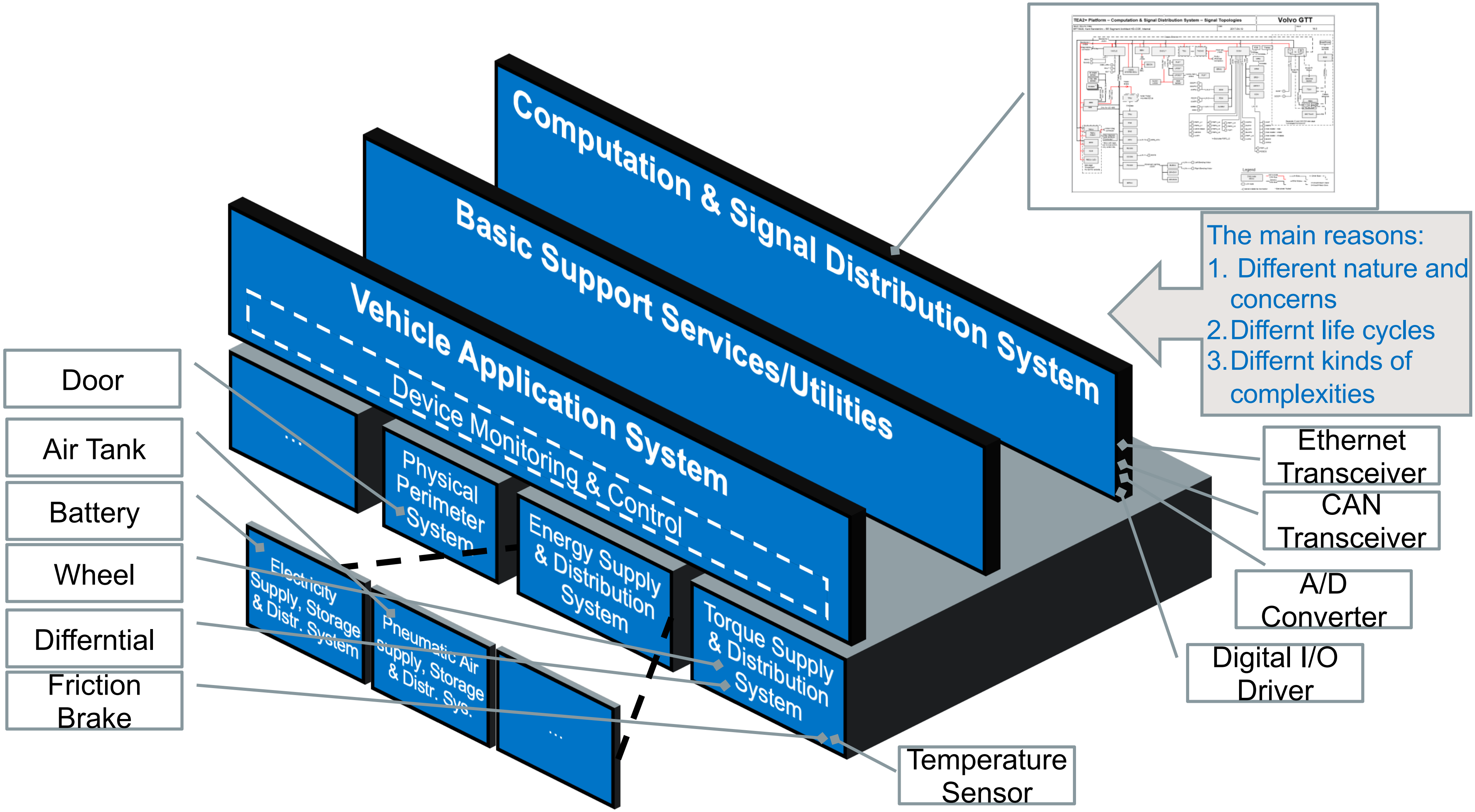
Layering (2)
Example: Communication Stack





Picture from Jeremy Bradbury, Queens Univ. Canada

An example in Automotive Domain:
Vehicle Monitoring & Control System


TEA2+ Platform – Computation & Signal Distribution System – Signal Topologies
Volvo GTT
Computation & Signal Distribution System
Basic Support Services/Utilities
Vehicle Application System
Device Monitoring & Control
...
Physical Perimeter System
Energy Supply & Distribution System
Torque Supply & Distribution System
Electricity Supply, Storage & Distr. System
Pneumatic Air Supply, Storage & Distr. Sys.
...
Door
Air Tank
Battery
Wheel
Differntial
Friction Brake
Ethernet Transceiver
CAN Transceiver
A/D Converter
Digital I/O Driver
Temperature Sensor

The main reasons:
1. Different nature and concerns
2. Differnt life cycles
3. Differnt kinds of complexities

Function: Monitoring Air Inlet Pressure

Converts look and feel into pixels

Deals with the look and feel of the user interface instrument

Deals with WHAT shall be shown to a driver

A representation of Inlet Air and its charactersitics, i.e. convert measurement to a pressure unit like kPa.

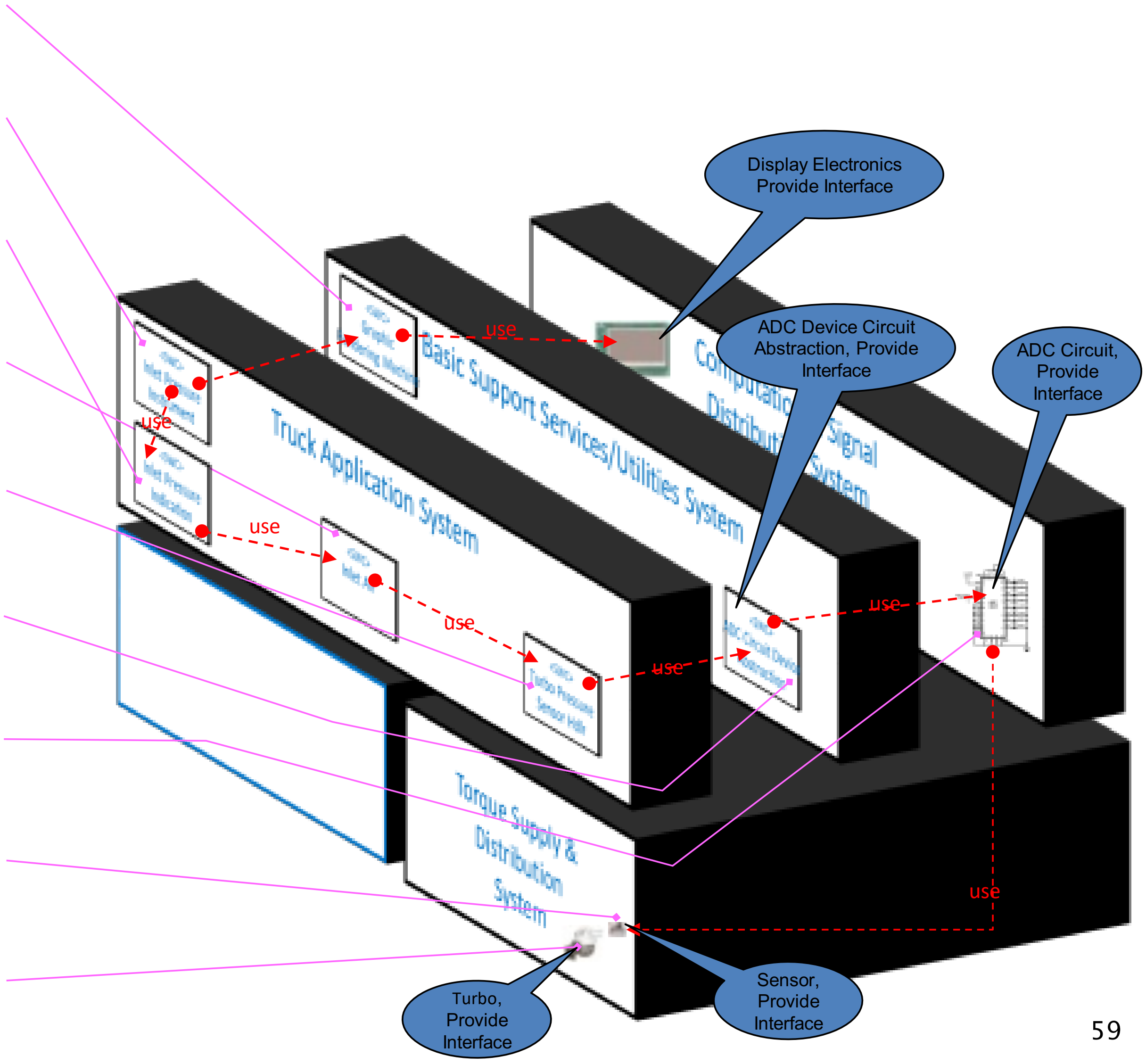
Deals with sensor characteristics, e.g. hysteresis, linerarity, …

Access the A/D circuit, i.e. deals with circuit characteristics

Convert an analogue value to a digital, change units

Convert a strain to a voltage level, change units

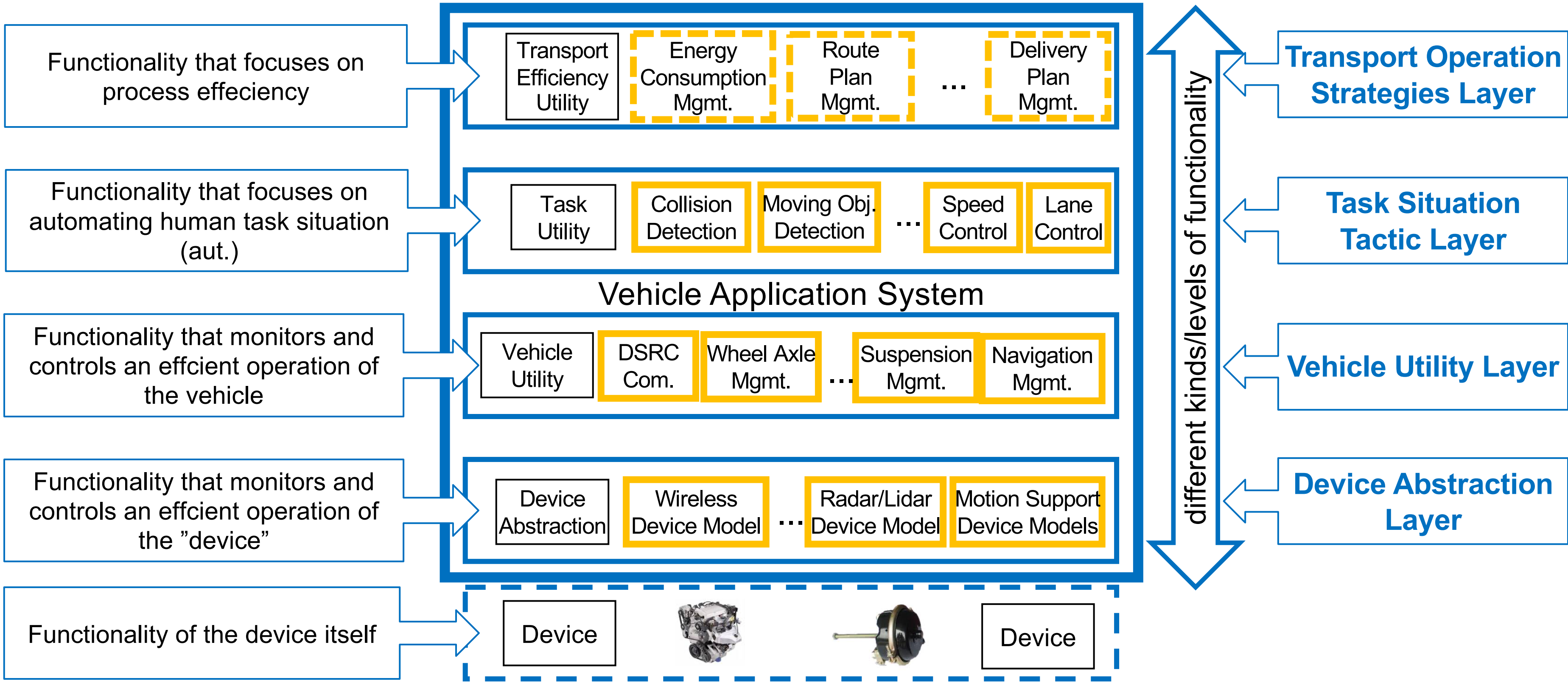
Increase pressure = change characteristics of the same unit



Display Electronics Provide Interface
ADC Device Circuit Abstraction, Provide Interface
ADC Circuit, Provide Interface
use
use
use
use
use
use
use
Basic Support Services/Utilities System
Truck Application System
Computatio
Distribut
Signal
System
Torque Supply & Distribution System
Turbo, Provide Interface
Sensor, Provide Interface

Layers in the Vehicle (Truck) Application Systems
Example Platooning Function

Example:
Platooning



Functionality that focuses on process effeciency

Functionality that focuses on automating human task situation (aut.)

Functionality that monitors and controls an effcient operation of the vehicle

Functionality that monitors and controls an effcient operation of the ”device”

Functionality of the device itself

Transport Efficiency Utility

Energy Consumption Mgmt.

Route Plan Mgmt.

…

Delivery Plan Mgmt.

Task Utility

Collision Detection

Moving Obj. Detection

…

Speed Control

Lane Control

Vehicle Application System

Vehicle Utility

DSRC Com.

Wheel Axle Mgmt.

…

Suspension Mgmt.

Navigation Mgmt.

Device Abstraction

Wireless Device Model

…

Radar/Lidar Device Model

Motion Support Device Models

Device

Device

different kinds/levels of functionality

Transport Operation Strategies Layer

Task Situation Tactic Layer

Vehicle Utility Layer

Device Abstraction Layer

Layering (4) Quality Factors

Scalability:	n.a.*
Flexibility:	layers can be redefined
Robustness:	‘weakest layer’ is limitation
Security:	security measures should be taken at every layers’ interface

To understand a system as a whole, the number of layers
Should be limited to an intellectually manageable number: ±7

Layering (5) Application Context

Rules of thumb for using layering:

- if data processing progresses through successive levels of abstraction
  (vertical composition of functionality)

Layering is a technique that helps in structuring systems

Typical examples: OS, device drivers, virtual machine (JVM), ISO,
Client/Server

Division of Functionality

Pipeline:

- Multiple functional units operating in sequence (units chosen as steps in process)
- Regular pattern of computation for the class of inputs
- Functional units at same level of abstraction

Blackboard:

- Multiple functional units where order of operation is irregular or not know a-priori
- Allows concurrent operation of functional units
- Functional units at same level of abstraction (typically highly specialized processing)

Layered:

- Functionality (services) which are concerned with same level of abstraction are grouped

Summary Architectural Styles

Every Architect should have a standard set of architectural styles in his/her repertoire

- it is important to understand the essential properties of each style: when to (not) use them
- examples:
  - C/S, pipe and filters, blackboard, pub/sub, P2P

The choice for a style can make a big difference in the quality properties of a system

- analysis of the differences can provide rational for choosing a style

Questions ?