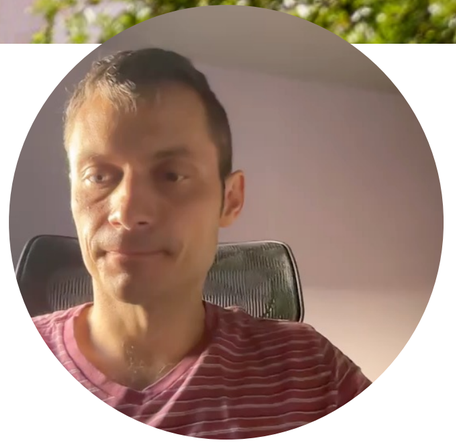
Architectural Styles



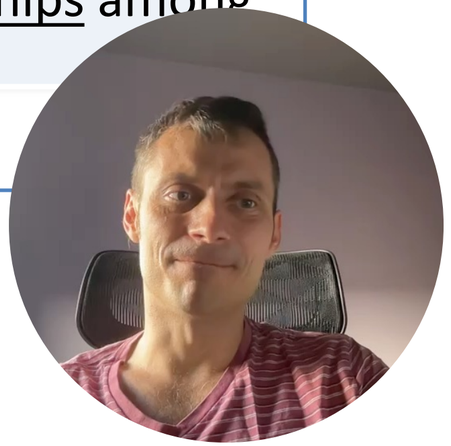
Software Architectures

- Software Architectures are usually complex
- Often, we reduce to an abstraction
- **Architectural Styles**
  - Layered style
- **Architectural Patterns**
  - Model View Controller

Architecture Definition [UML/Bass]

Architecture is the organizational **structure** and associated **behavior** of a system. An architecture can be recursively decomposed into parts that interact through interfaces, relationships that connect parts, and constraints for assembling parts. Parts that interact through interfaces include classes, components and subsystems.

The software architecture of a program or computing system is the **structure** or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.



Style, reference model a reference architecture

- Defining the following terms

1. architectural style
2. reference model
3. reference architecture

Architectural style

- Describes:
  - Types of **components**,
  - Patterns for the **interaction** of components in terms of
    - Control flow x Data flow
- Understood as a set of constraints.
  - These constraints define a class of architectures.

Architecture style

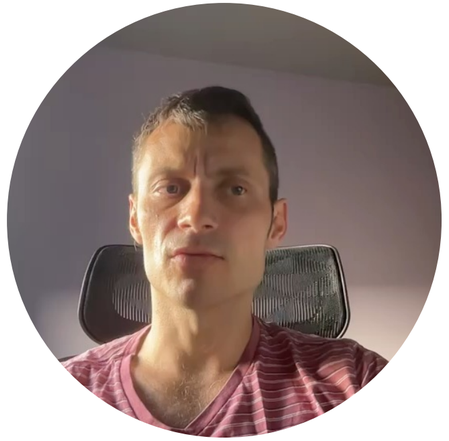
Reference model

Maps **functionality** with data flow onto **components**.

- Divides the problem domain
  - Does not provide any concrete software solution
- It is typically influenced by a given domain.

Example:

- Describe the properties of a plane/car



Reference model

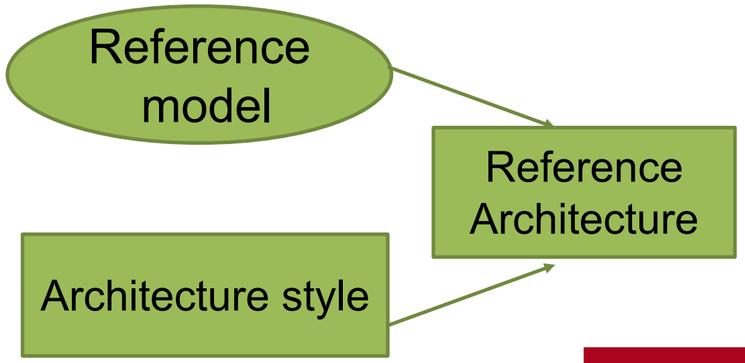
Architecture style

Reference architecture

- Reference model mapped onto software components, data flow among the components.

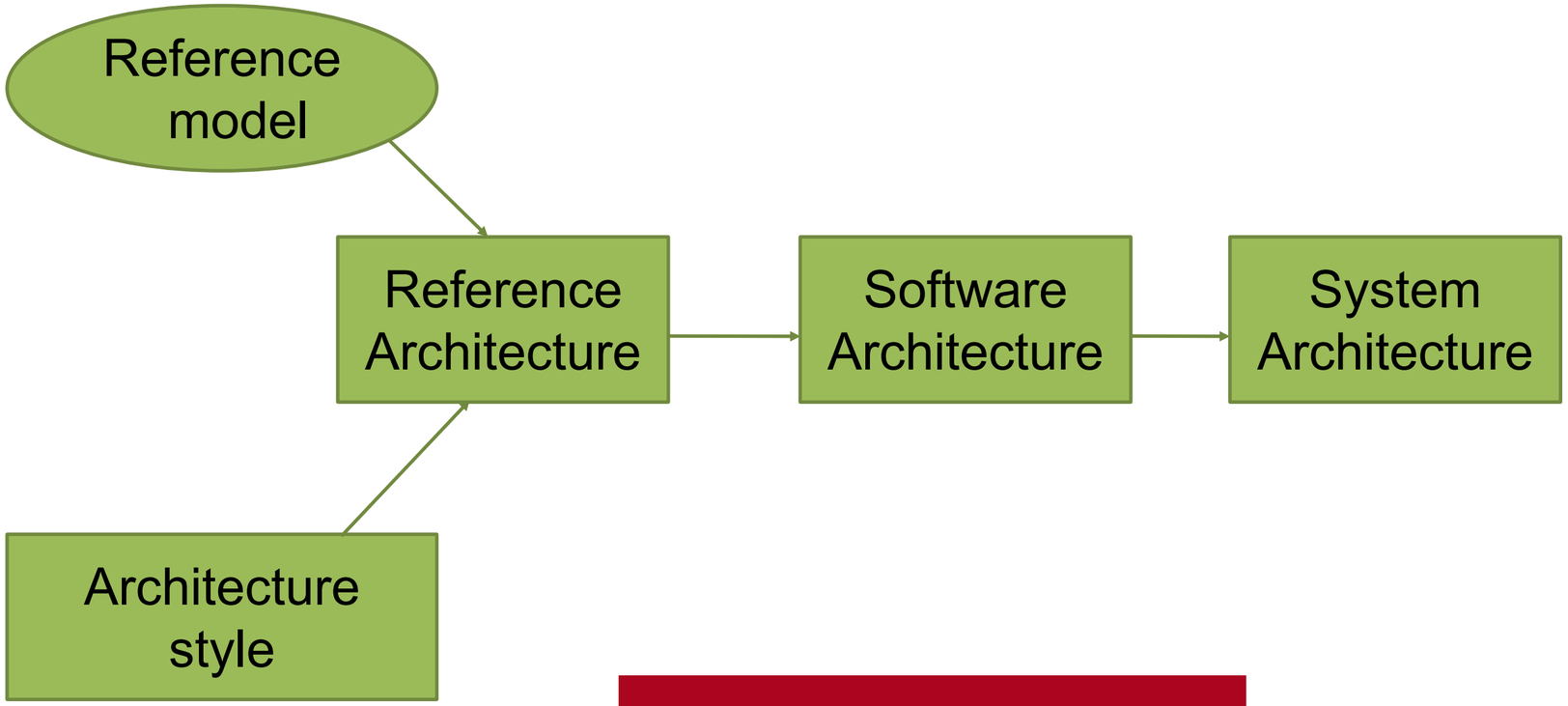
Ref. model divides functionality.
Ref. Architecture = maps functionality on decomposed system

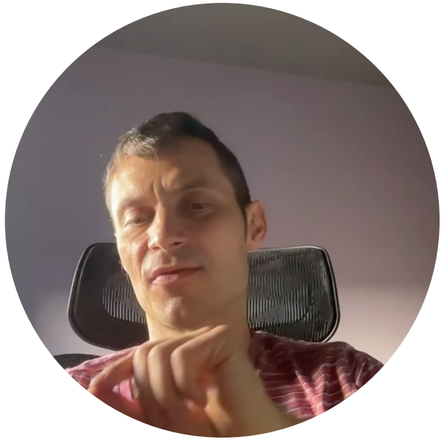
A component can implement part of functionality or many functionalities.



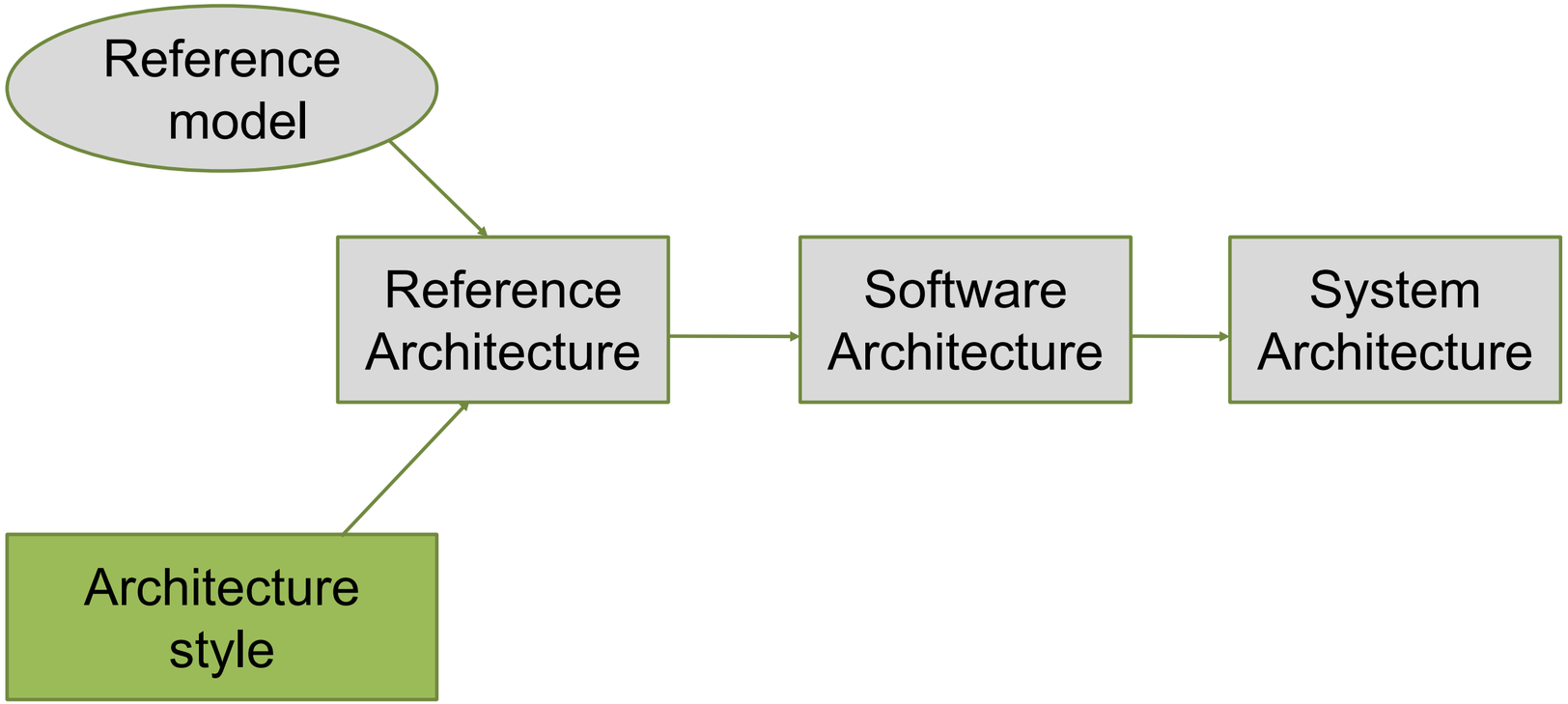
Long path to actual system architecture

- Reference model
- Architecture styles (a catalog of options)
- Reference Architecture
  - Not concrete architectures, but abstract step towards them





Architectural Styles



Architecture Styles

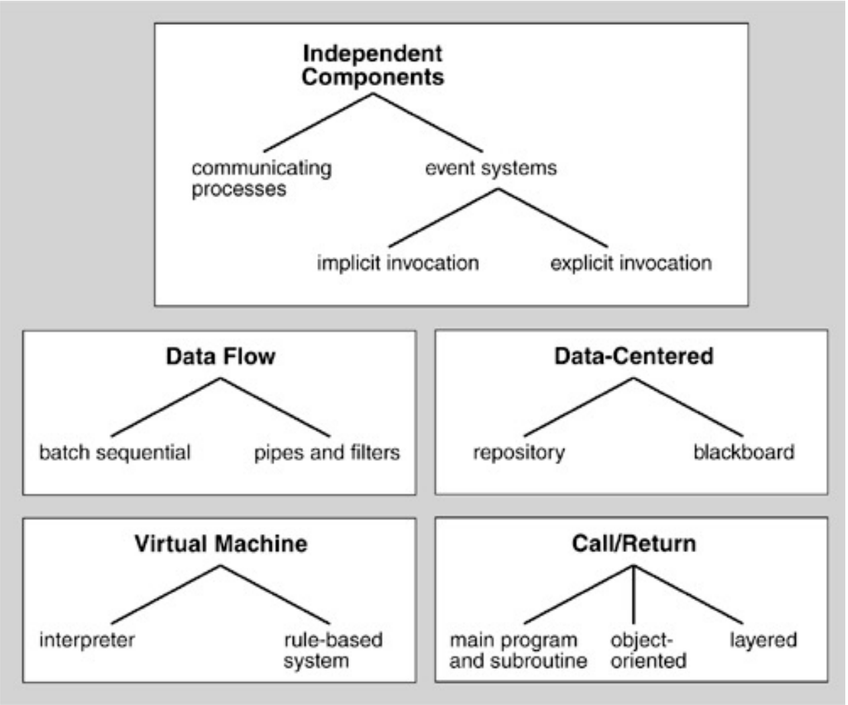
- Analogy
  - Gothics
  - Renaissance
  - Baroque
  - Antic style

- Key features and characteristics, rules for combination, however
- 2 or 200 windows and still a Gothics style
- Ambiguous

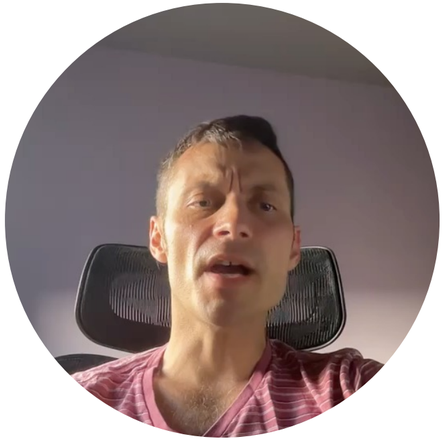
Architecture Styles

- A Framework that contains:
  - **Components**
    - Such as Process/Procedure/Object/..
  - **Topology structure**
    - Star, one-way flow
  - **Connectors**
    - Method call/data stream/Events/Sockets
  - **Constraints**

Architecture Styles



- Architectural styles do not have exact boundary
  - Systems can be classified by various styles
- Style catalogue to provide good overview of options
  - Similar QA, categorization, quick understanding

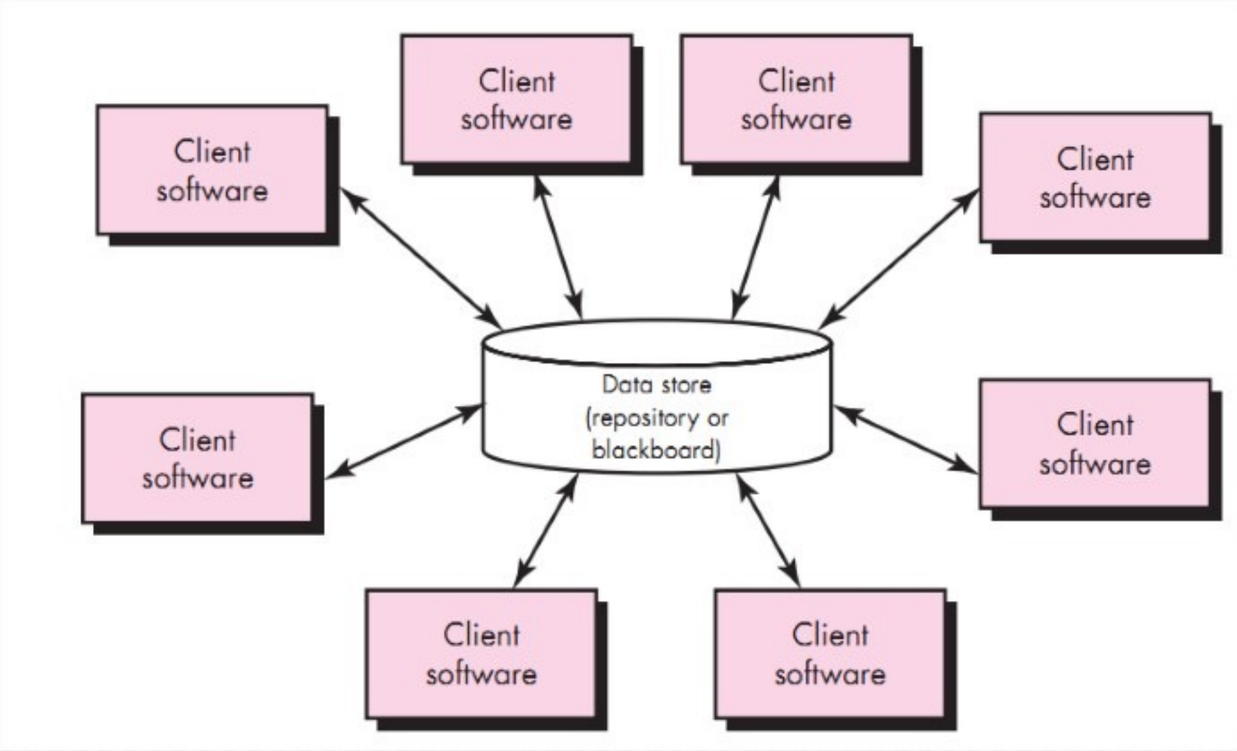


Architecture Styles

- **Data-centric**
  - Databases/blackboards
- **Data flow**
  - Pipe and Filter - data flow
- **Call and return**
  - Procedures/OOP/AOP
- **Layers**
- **Implicit invocation**
  - Events
- **Independent components**
  - Peer-to-peer
- **Virtual Machines**

} Separation of concerns

Data centric style



- **Data integration**
  - Data storage
  - Access to data

- Client vs. Data structure
  - Independent processes, own control
  - Data structure
    - Accessed by many clients
    - Global state
    - ACID
    - Passive vs Active

https://medium.com/

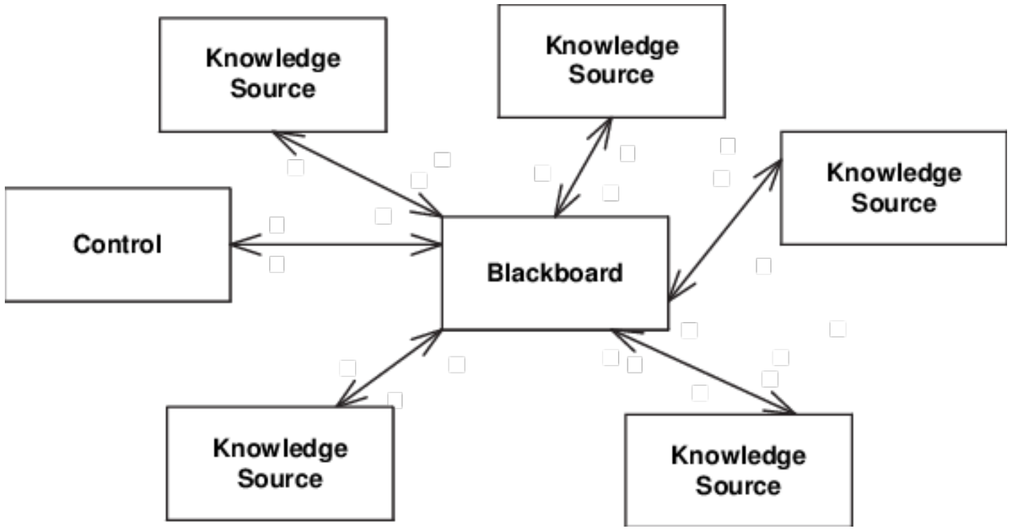
Data centric style

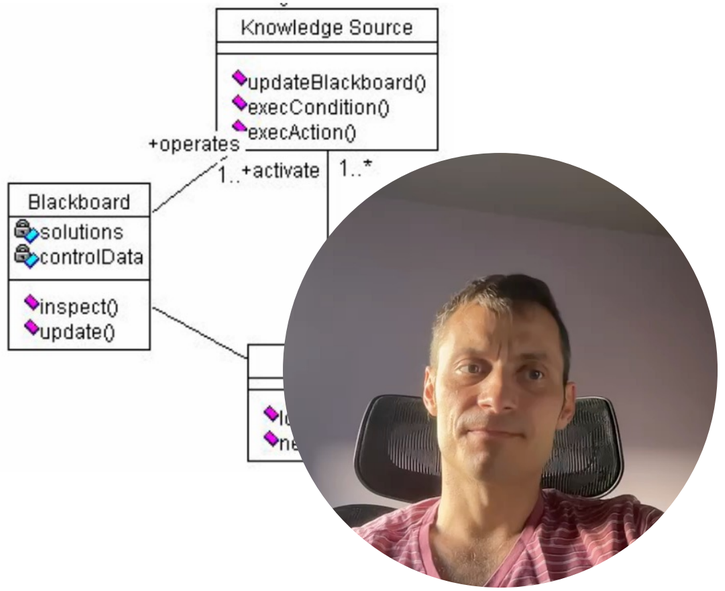
- **Passive**
  - Client accesses central data structure and modifies it
  - Central structure gives simultaneous access
  - Probes
  - Coordination by client

- **Active**
  - Notification mechanism
    - – inform me of changes - subscribers
  - Hierarchical structure
  - Bidirectional coordination
    - Language processing

Data centric style

- Repository
  - Database
  - Probe-echo
  - Client-drives communications
- Active Blackboard
  - Voice recognition
  - Compilers
  - Scheduler/controller

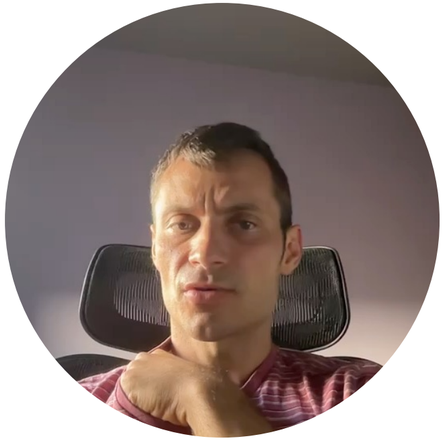




A **blackboard system** is an artificial intelligence approach

Data flow style

- Reusability and modifiability
- System is a list of transformations
  - Data flow by stages
- Types
  - Batch sequential
  - Pipes and filters



Data flow – batch processing

step1 < input > intermediate step2
< intermediate > output

- **Batch processing** is the execution of a series of jobs in a program on a computer without manual intervention
- Independent programs
- Run in sequence; wait on the previous process

step1 < input | step2 > output

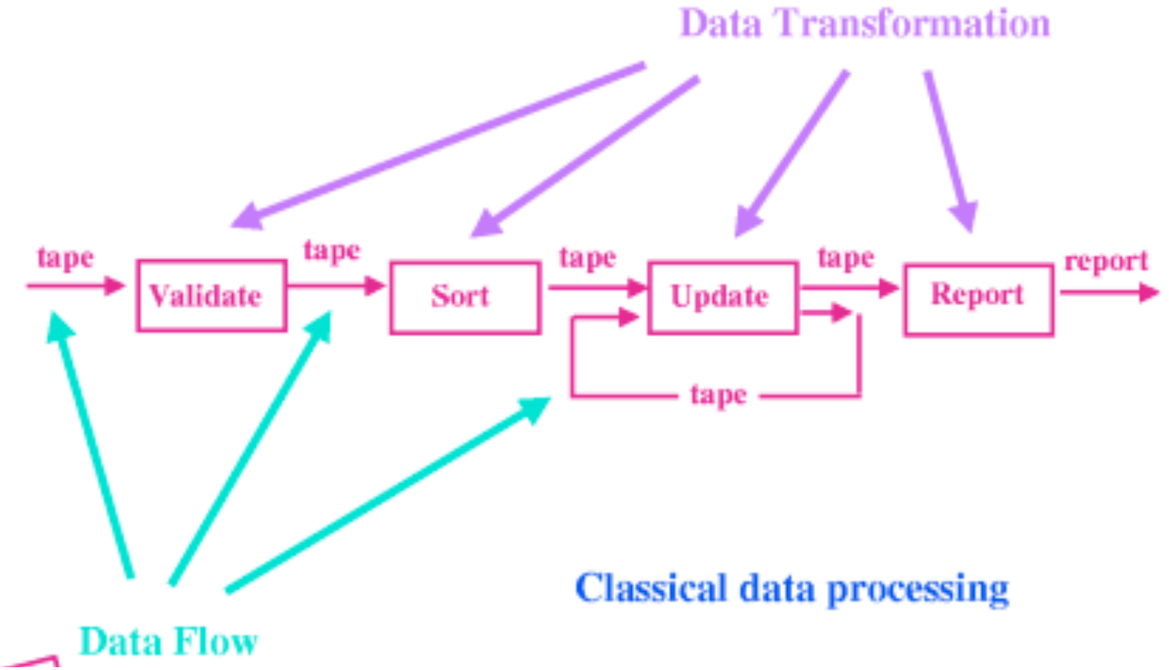
JBatch is the standardized **batch processing** framework for the Java EE platform. It eases tedious w
on **batch** programming such as transaction management of bulk **processing**, parallel **processing**, flow c

Data flow – pipes and filters

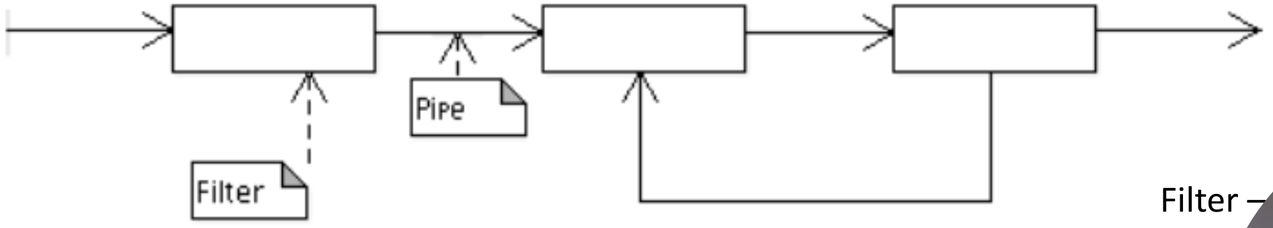
- Run in sequence, **do not** wait on previous, process existing outcome
- Pipeline
- Components
  - Pipes – not work just transfer – no state
  - Filters – worker
    - No interaction among filters
- Incremental transformation

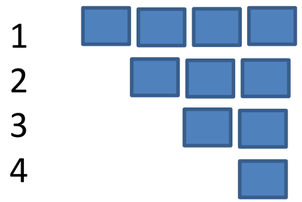
Data flow

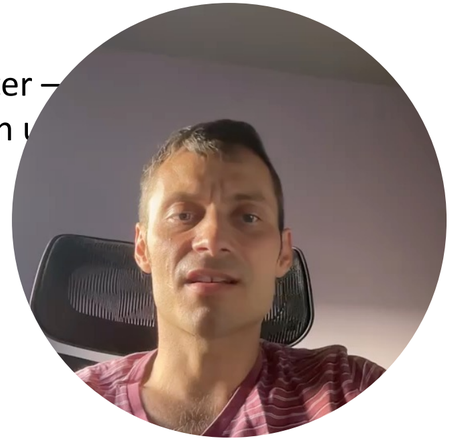
- Batch processing



- Pipes n filters



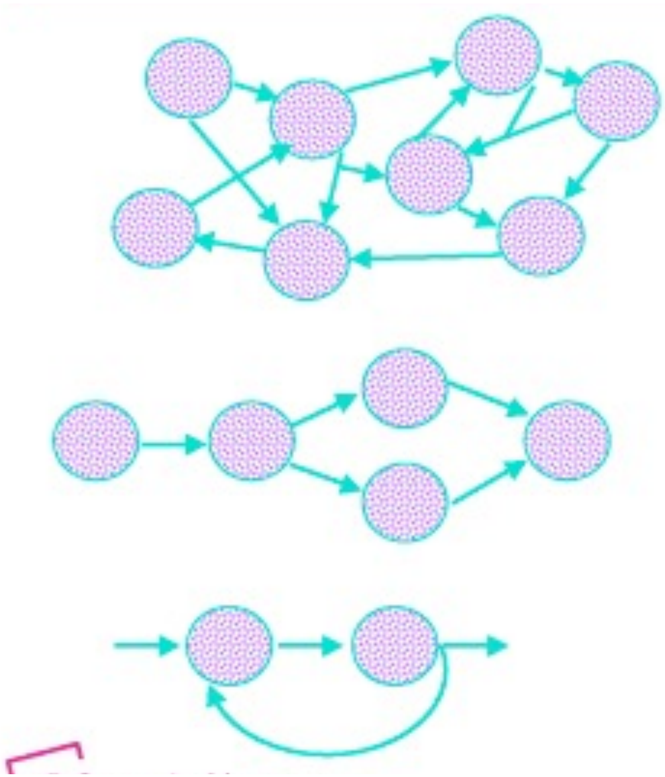




Filter –
Run

Data flow

- General data-flow



In general, data can flow in arbitrary patterns

Here we are primarily interested in nearly- linear data flow systems,

or in very simple, highly constrained cyclic structures

Call and Return (C&R)

- Procedural
- **OOD**
- Remote-procedure call
- Aspect-oriented programming

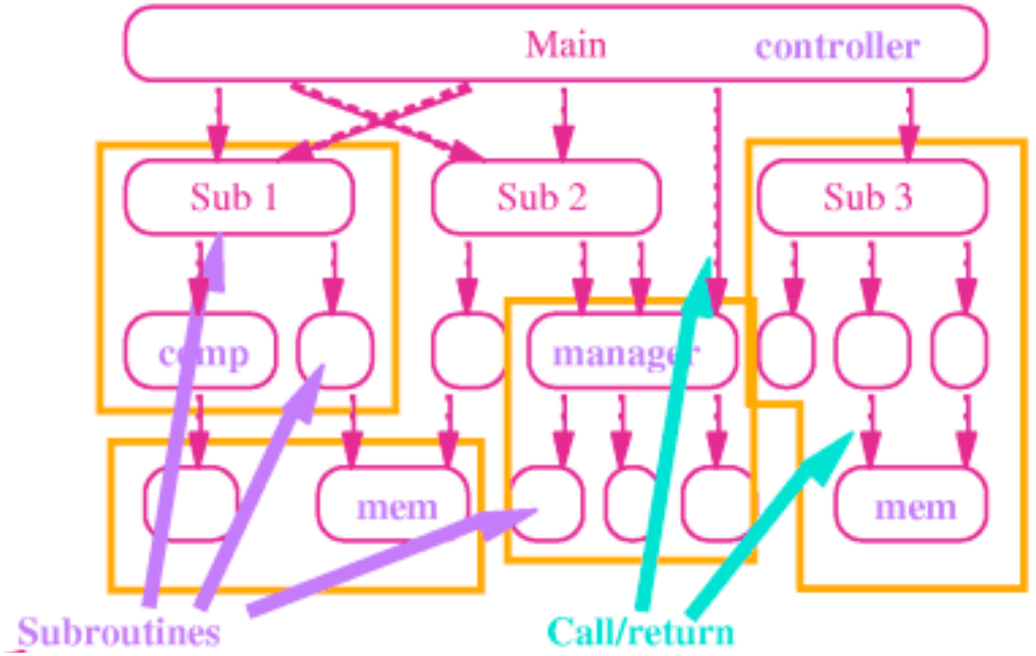
Call and Return (C&R)

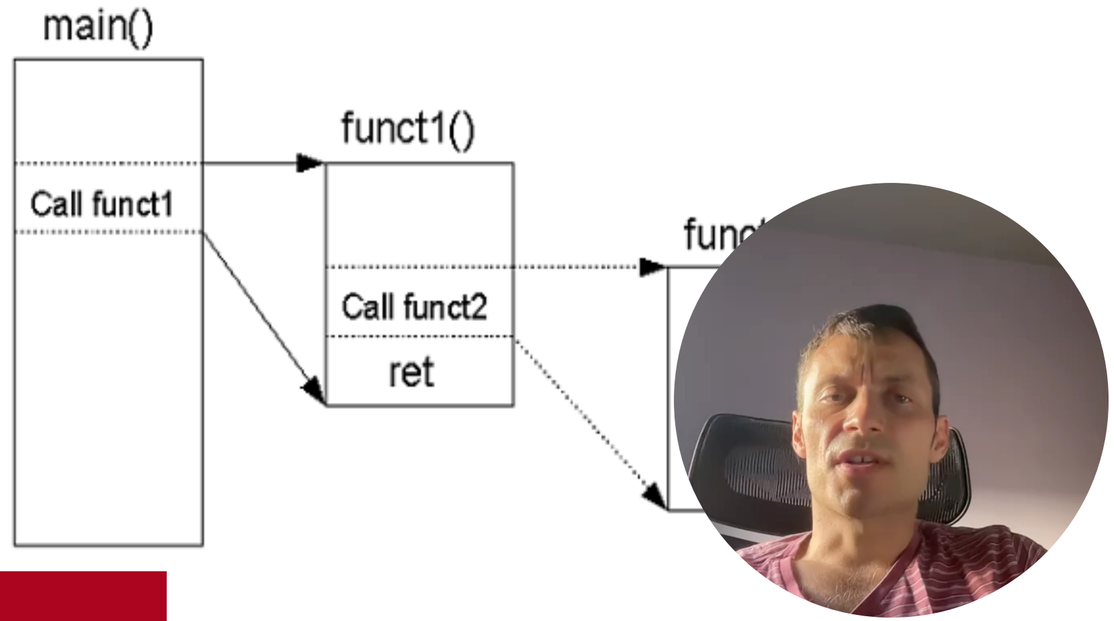
- **Modifiability, readability, scaling**
- **Dominant approach for decades**

(C&R) Procedures

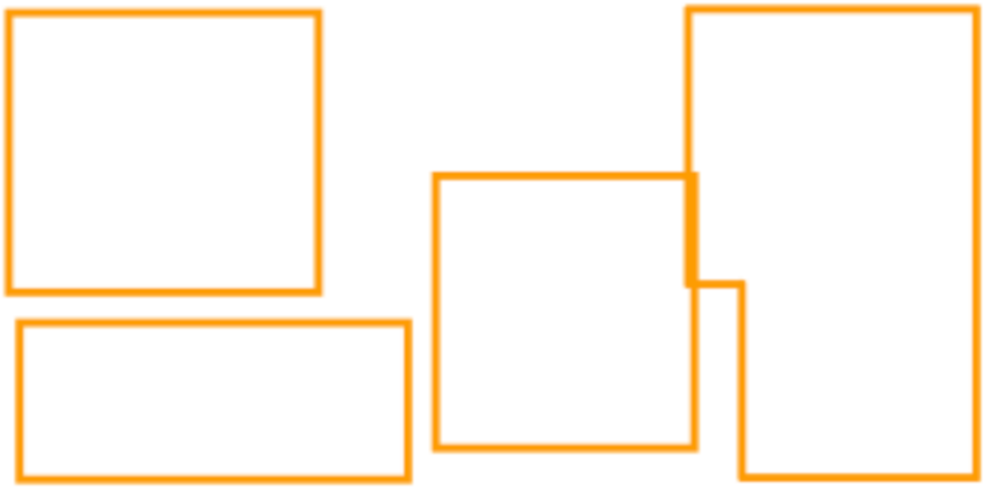
- **Recursion**
- **Parameters**

  - **Functional decomposition**
  - **Main thread in the maze**





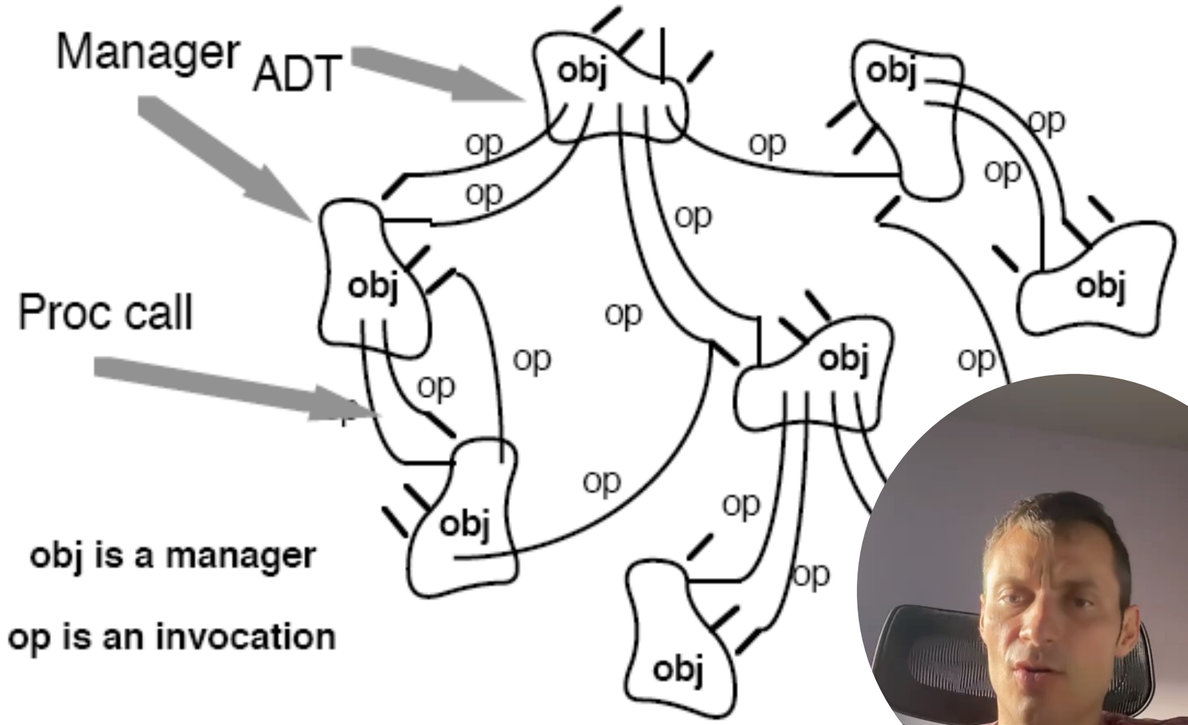
(C&R) Procedures



- **Recursion**
- **Parameters**
  - **Functional decomposition**
  - **Main thread in the maze**

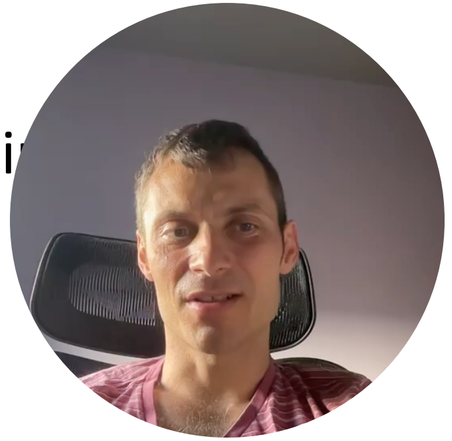
(C&R) Object-Oriented style

- Component
  - Object
    - Inf. Hiding
    - Encapsulation
    - Polymorphism
- Predecessor is ATD
  - Abstract data type
  - No polymorphism
- Separation of concerns



(C&R) Object-Oriented style

- Separation of concerns (Soc)
  - a design principle used in programming separates a program into units, with minimal overlapping between the “functions” (concerns) of the individual units. The concerns are separated using modularization, encapsulation, and arrangement in software layers.
  - A concern can be as general as "the details of the hardware for an application", or as specific as "the name of which class to instantiate.”
    - Security, performance, privacy, encapsulating knowledge of a product, transaction, logging,…
  - A program that embodies SoC well is called a modular program.
  - Modularity, and hence SoC, is achieved by encapsulating information i section of code that has a well-defined interface
  - Encapsulation is a means of information hiding

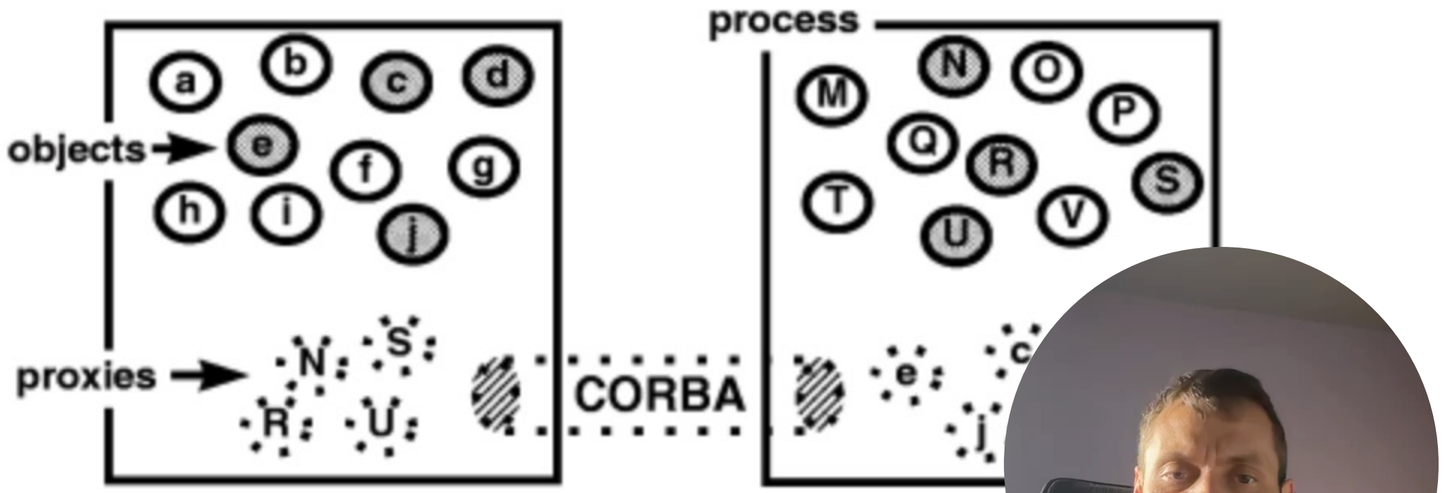


(C&R) Remote Procedure Call

- **Sub-method**
  - **Different location**
  - **Performance**
  - **Distributed systems**
  - **Location transparency**

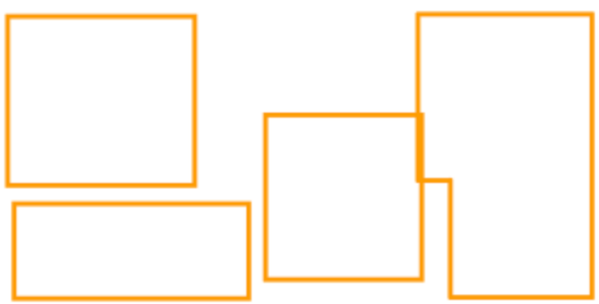
(C&R) Remote Procedure Call + Objects

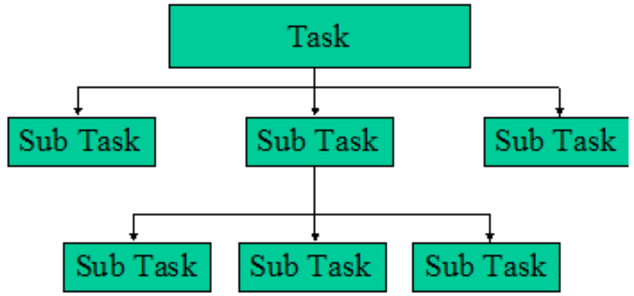
- **CORBA**
  - **Common Object Request Broker Architecture**



(C&R) Aspect Oriented Programming

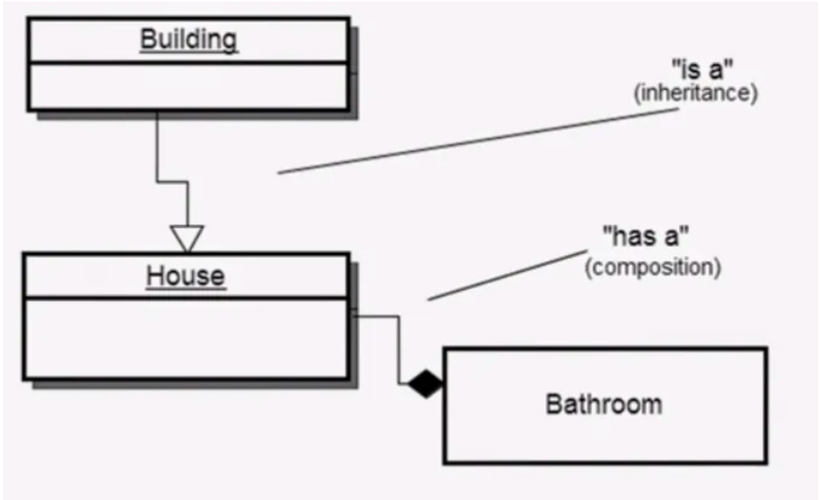
- **Procedures / Objects**

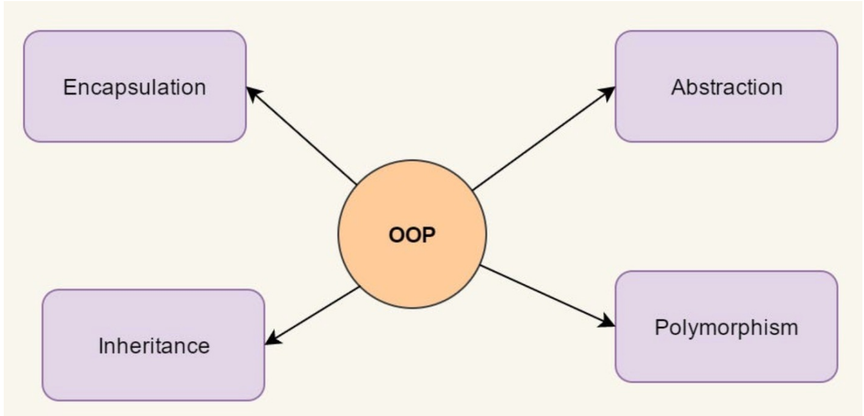




- **Functional decomposition**

- **Object-Oriented decomposition**
  - Composition + Inheritance

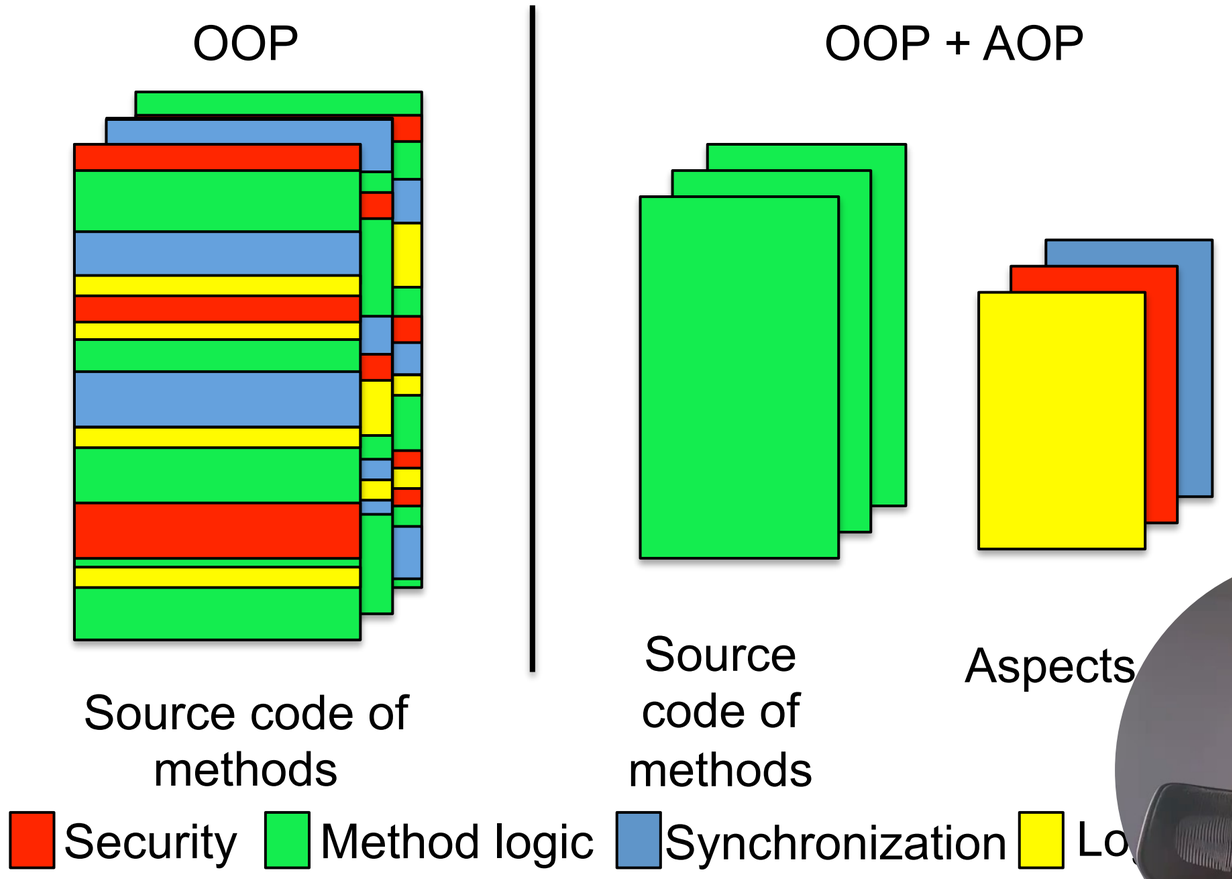


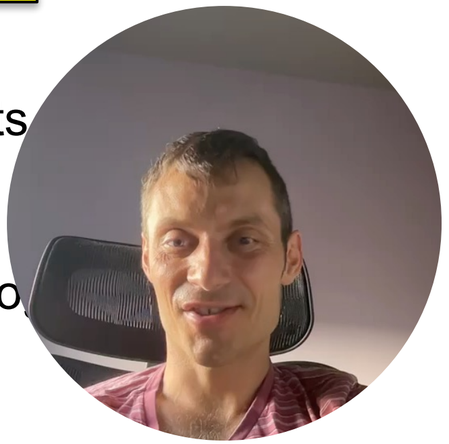


https://medium.com/

(C&R) Aspect Oriented Programming

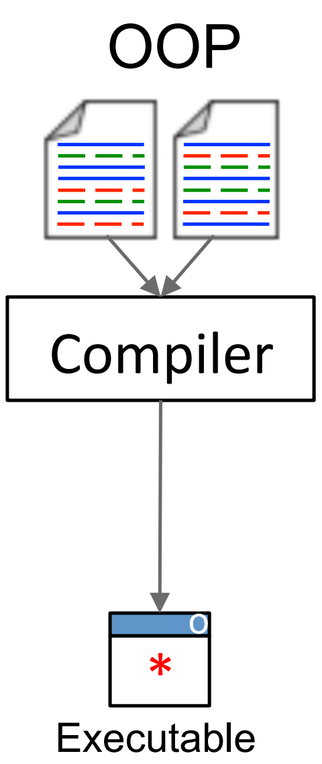
- Component + Aspect
  - Join point
  - Pointcut/Advice
  - Aspect weaver
- Separation of
  - **crosscutting**
  - concerns

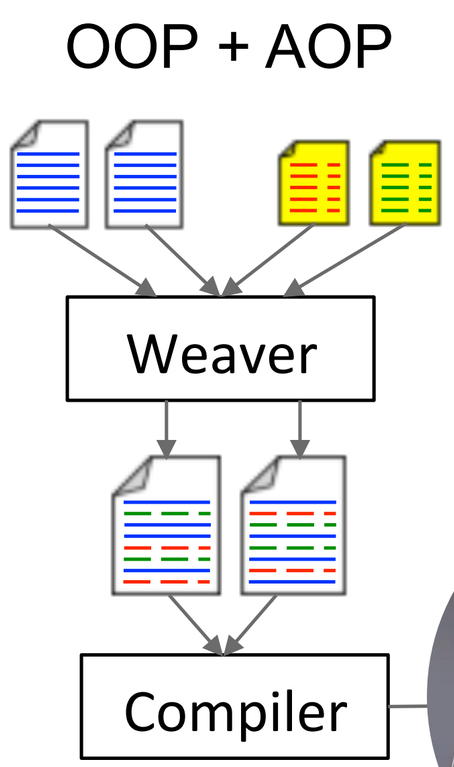




(C&R) Aspect Oriented Programming

- Component + Aspect
  - *Join point*
  - *Pointcut/Advice*
  - *Aspect weaver*

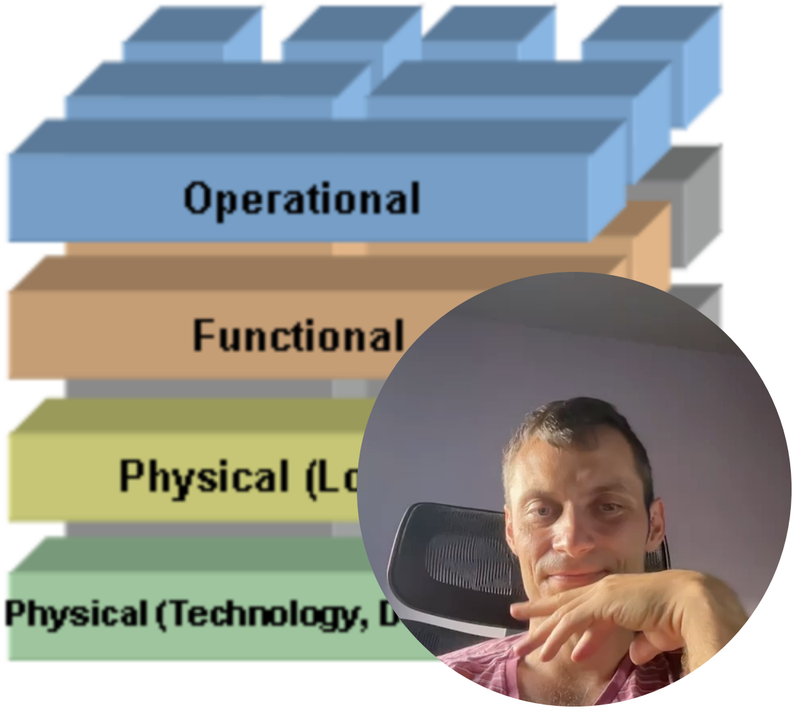




(C&R) Aspect Oriented Programming

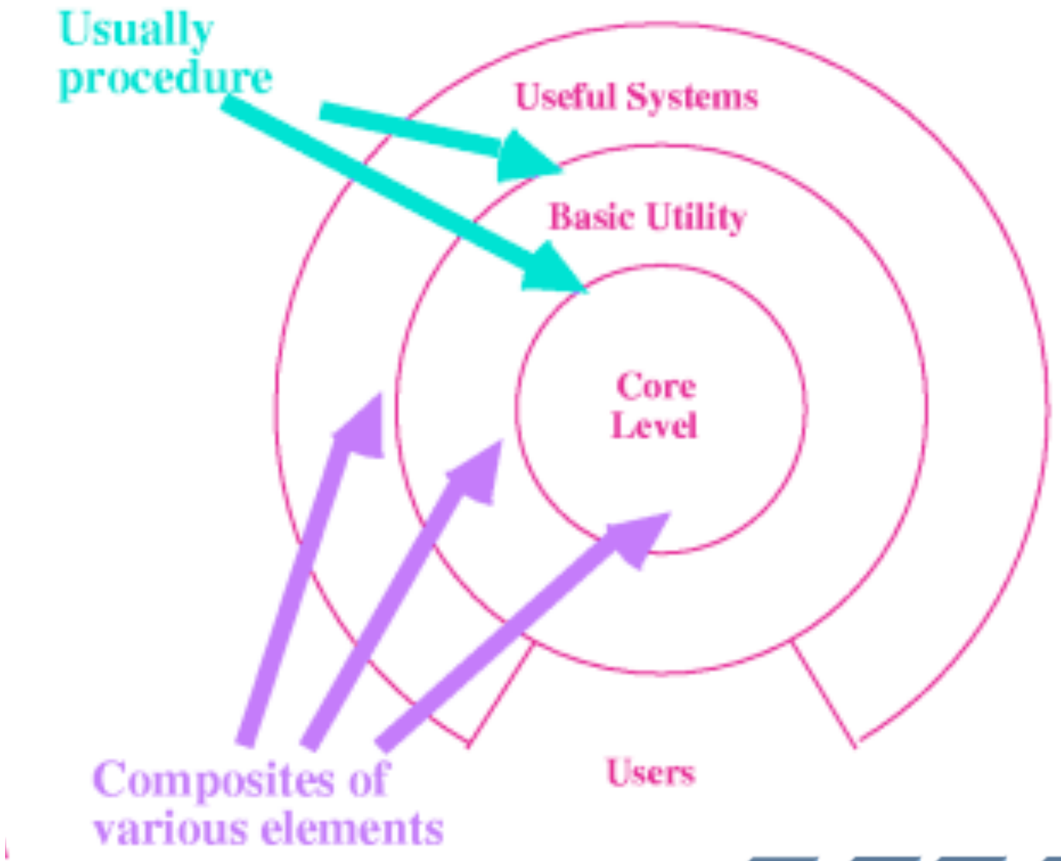
Layers

- Layered designs in information systems are another embodiment of the separation of concerns (e.g., presentation layer, business logic layer, data access layer, persistence layer).



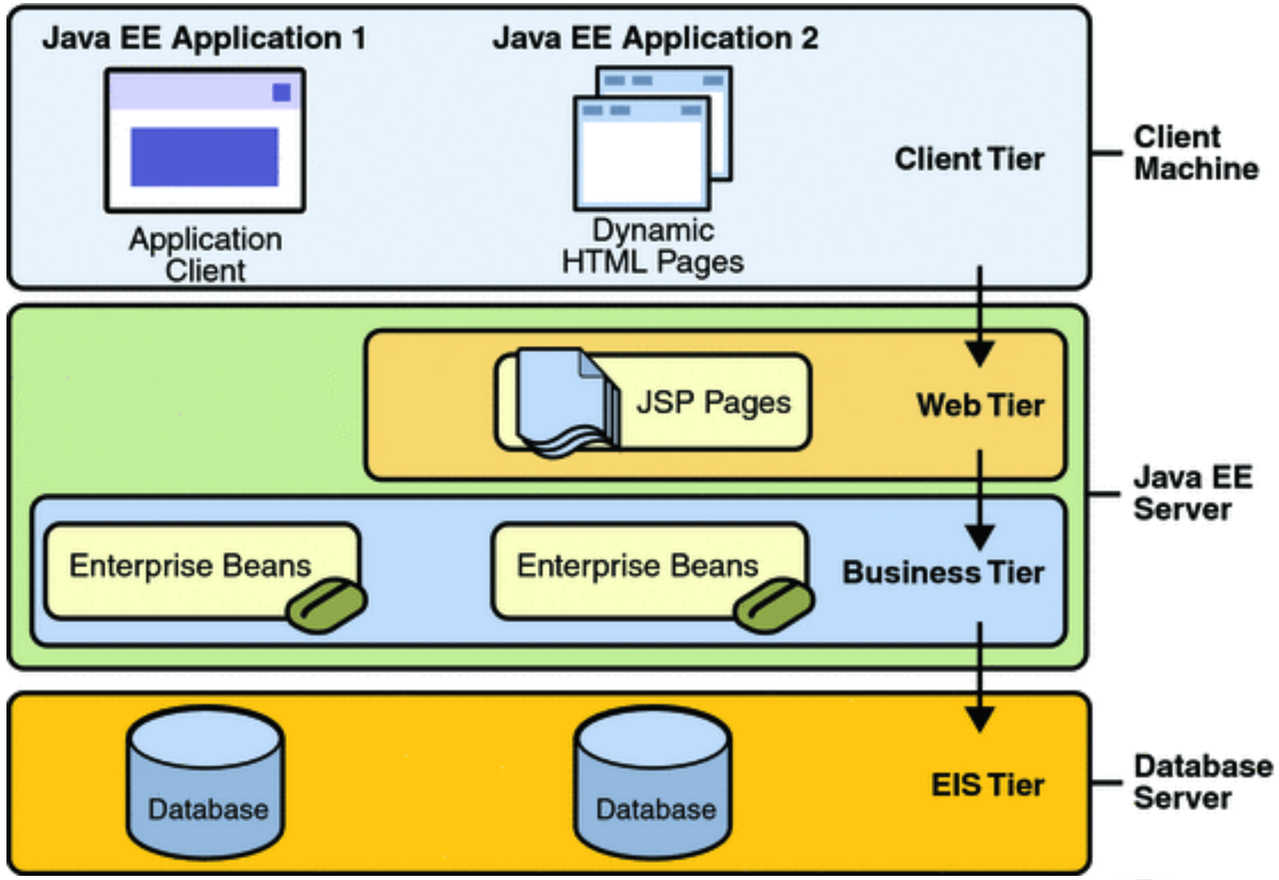
Layers

- Strict/free calls across layers
  - Must call the closest layer?
- Portability
- Java EE/ISO OSI
- Extension of core
- Microkernel



Layers

- Java EE
  - Tiers

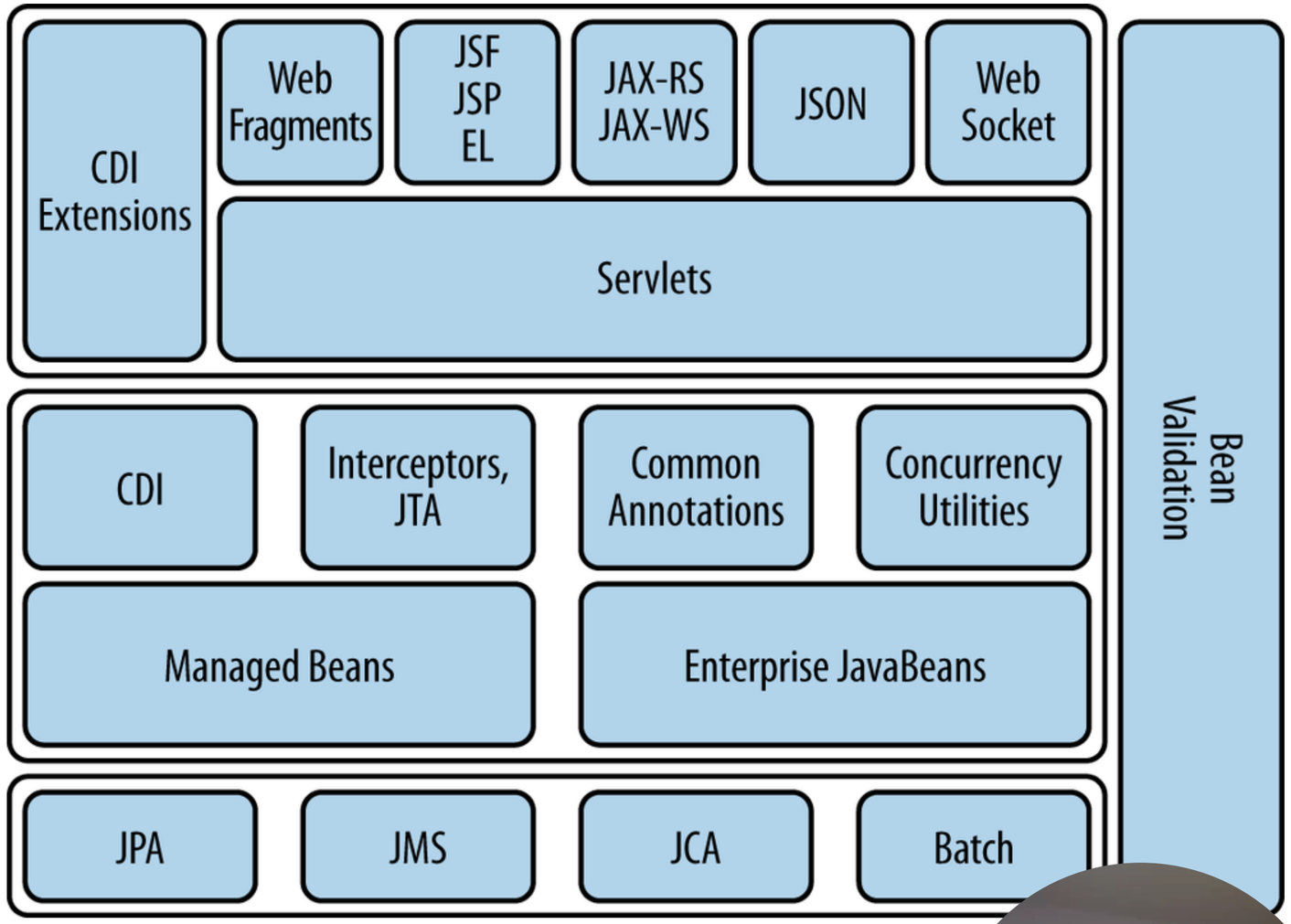


https://docs.oracle.co

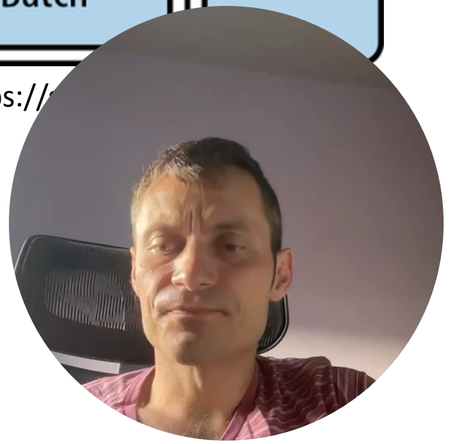
- Tier vs. layer
  - A 'layer' refers to a functional division of the software, but a 'tier' refers to a functional division of th that runs on infrastructure separate from the other divisions.

Layers

- Java EE framework

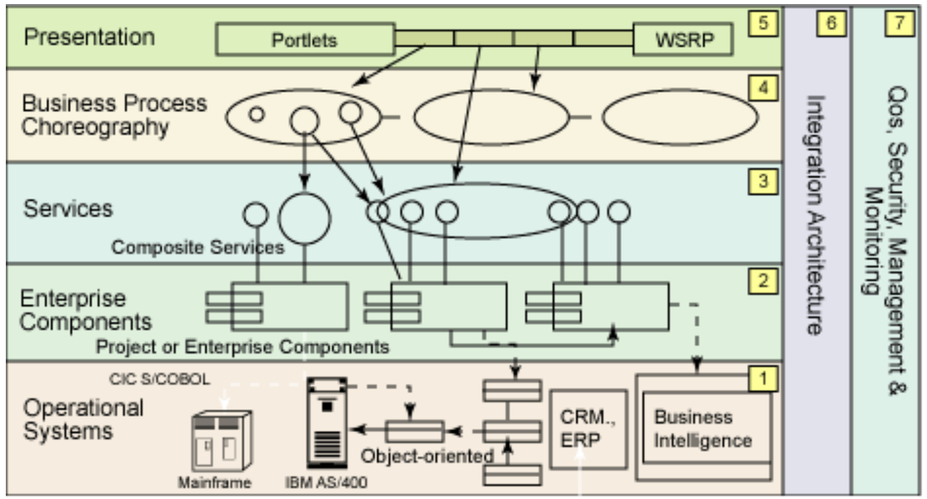


https://



Layers

- Enterprise Architecture

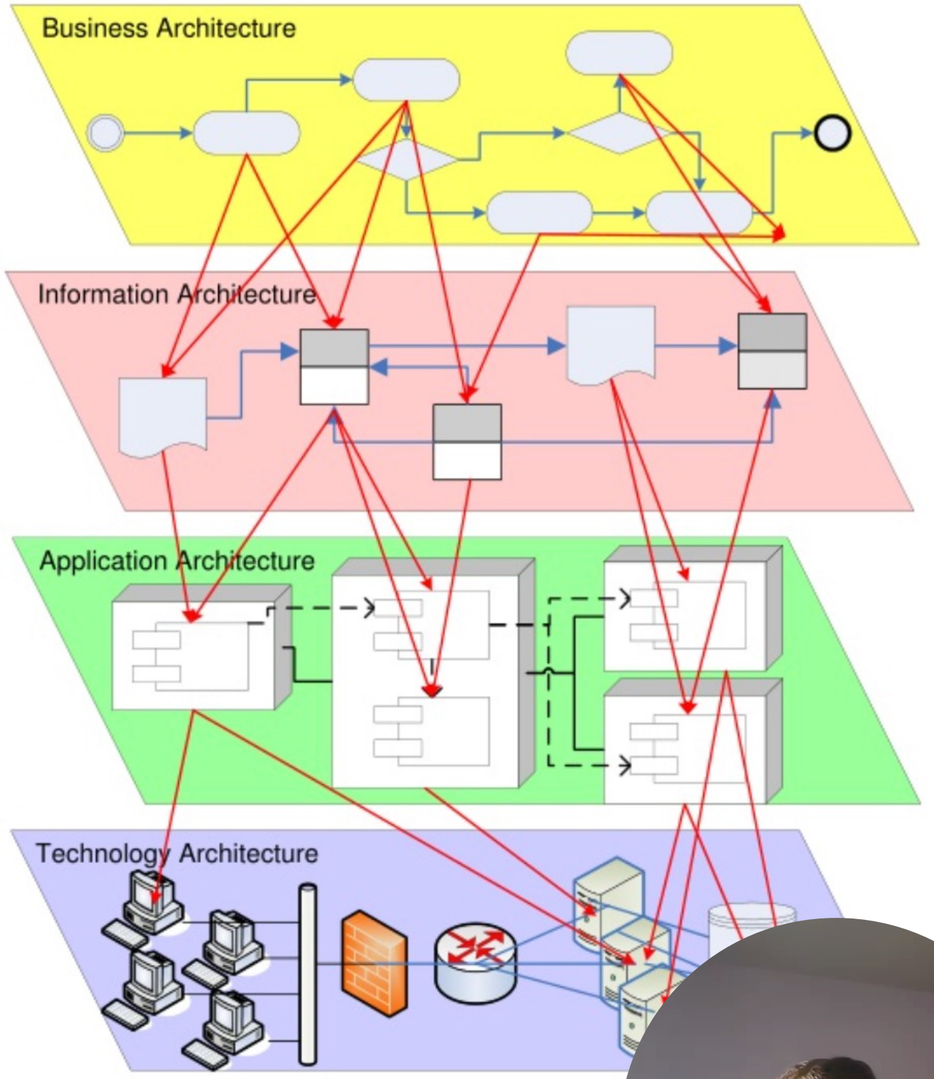


Business (Process) Architecture:
- Business strategy, governance
- Key business processes
- Actors, organization

Information Architecture:
- Information object
- Logical and physical data
- Data management
- Metadata

Applications Architecture:
- Application portfolio
- Functional services
- Integration
- Application platforms

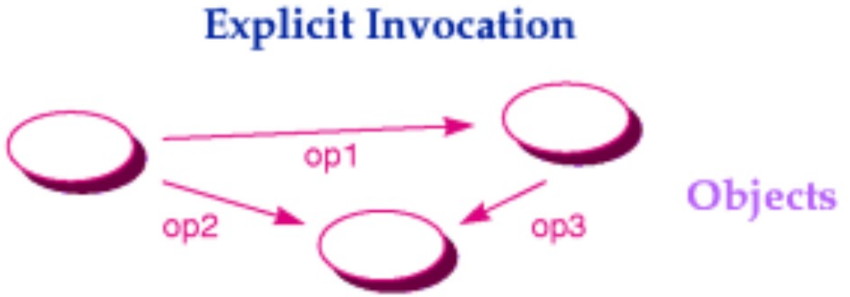
Technology Architecture:
- Databases, middleware
- Servers, workstations
- Networking
- Security solutions



Events & Implicit invocation

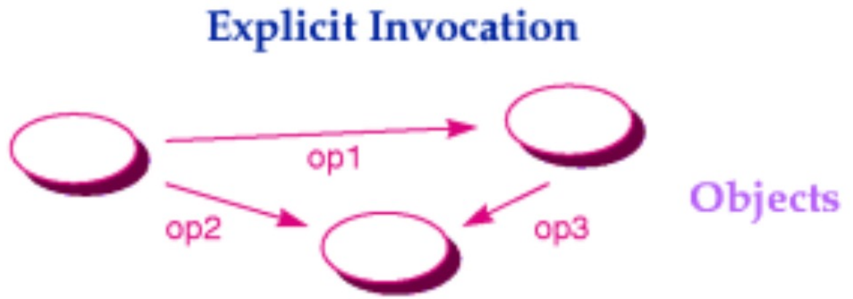
- Explicit vs. Implicit invocation
  - Events
    - Observer
    - Mediator

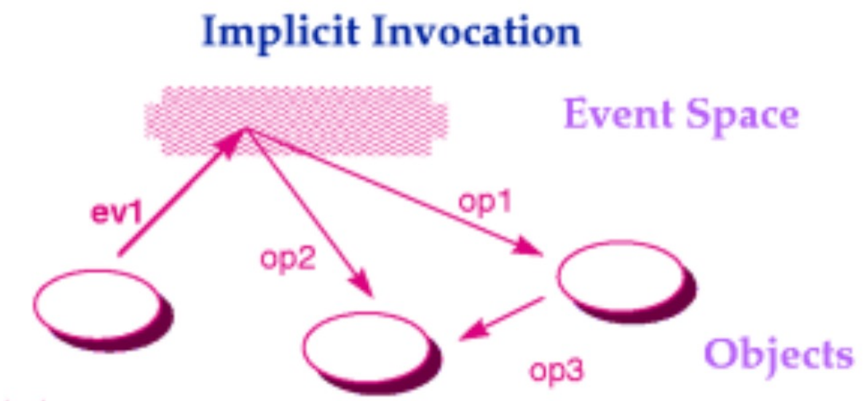
  - Modifiability
  - Integration
  - Performance



Events & Implicit invocation

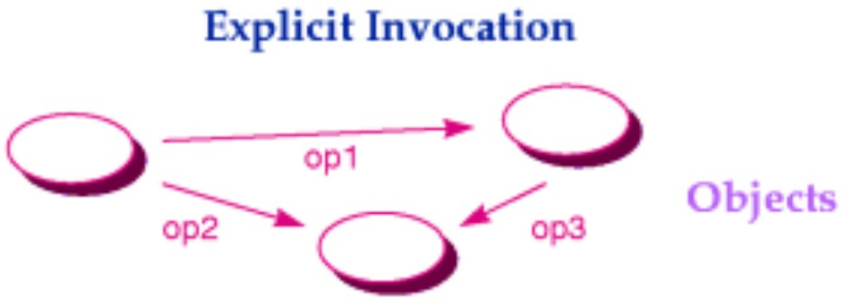
- Explicit vs. Implicit invocation
  - Events
    - Observer
    - Mediator
  - Modifiability
  - Integration
  - Performance

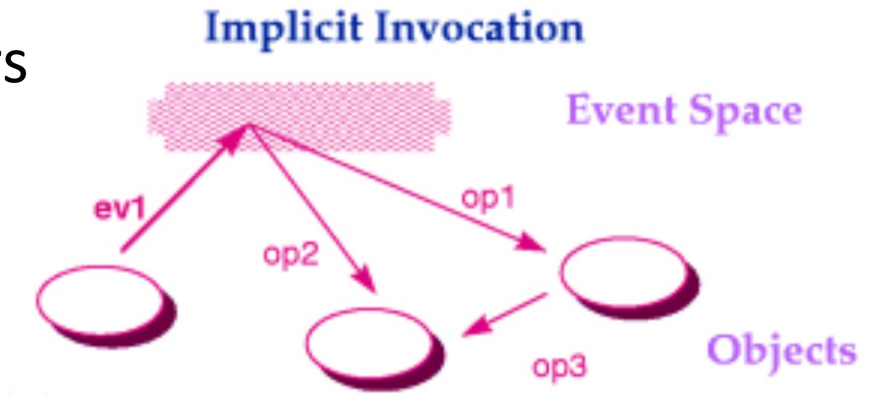


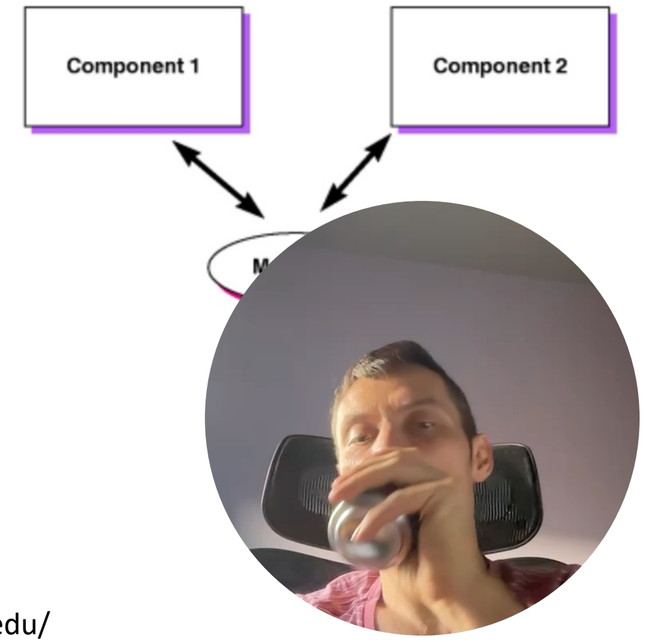


Events & Implicit invocation

- Consequences
  - Events
  - Publisher
  - Receiver
- Unknown senders or receivers
- Message manager

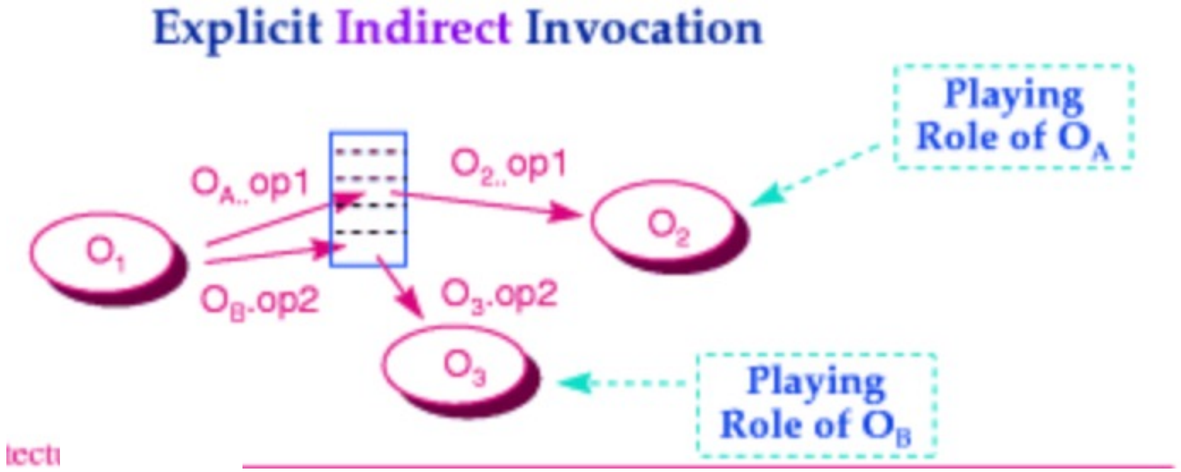






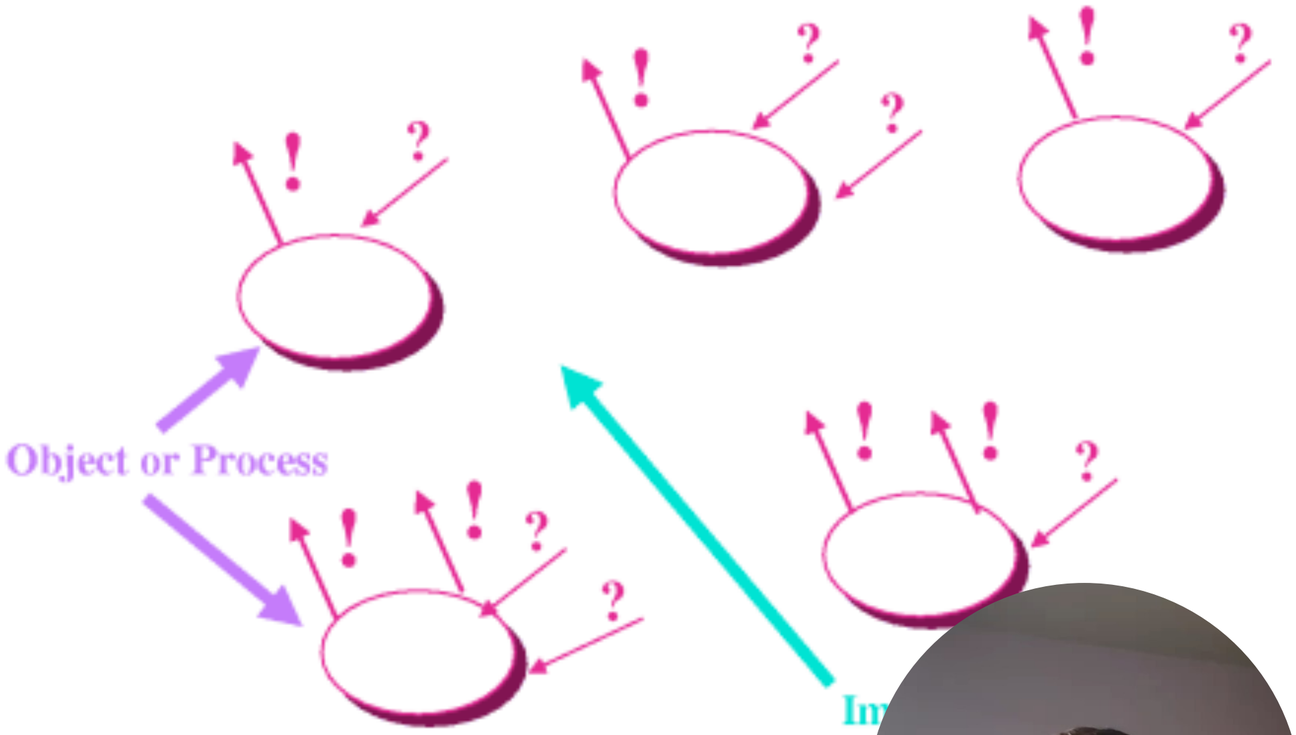
Events & Implicit invocation

- **Indirect** vs. Implicit invocation
  - Not the same



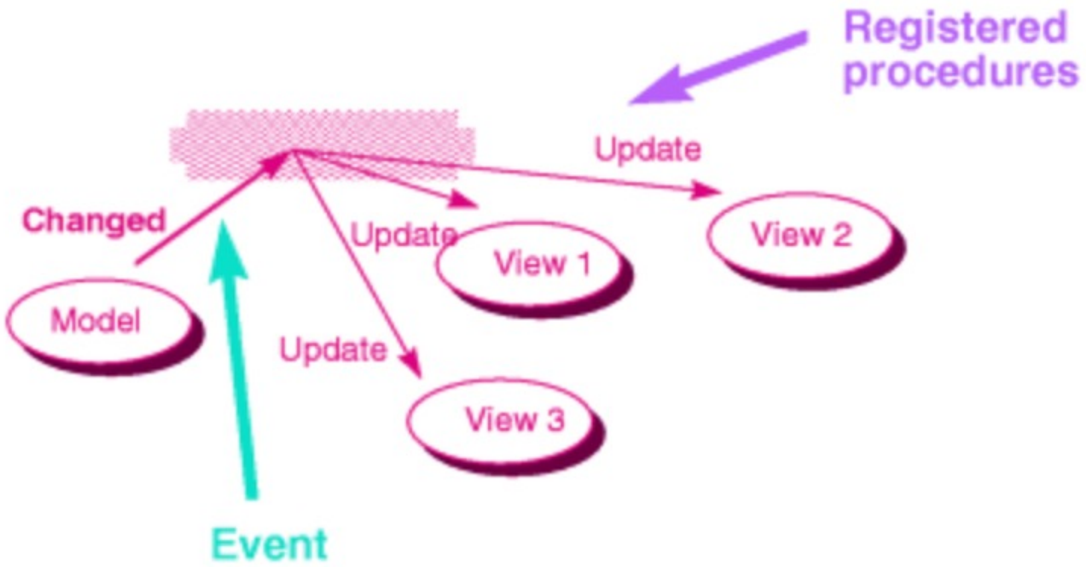
Events & Implicit invocation

- Event system
  - Components – process
    - Interface
      - Incoming/outgoing events
    - Connections
      - Callback for events
      - When an event is announced
        - Callback is invoked
      - Order of invocation non-deterministic



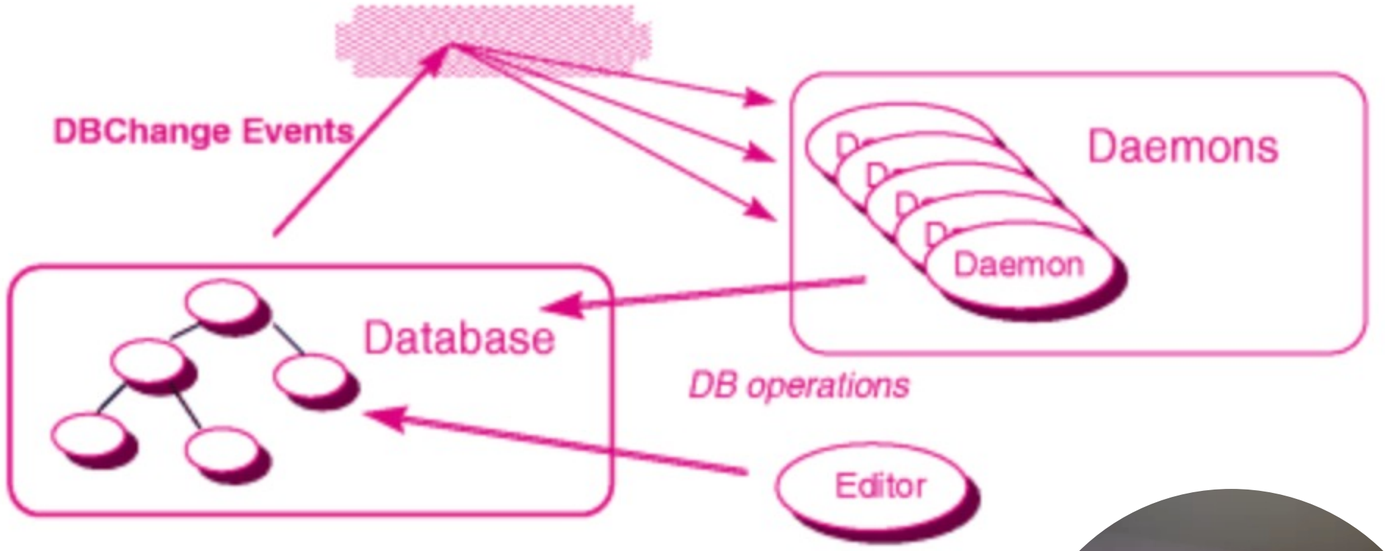
Events & Implicit invocation

- Event system
  - Components – process
    - Interface
      - Incoming/outgoing events
    - Connections
      - Callback for events
      - When an event is announced
        - Callback is invoked
      - Order of invocation non-deterministic



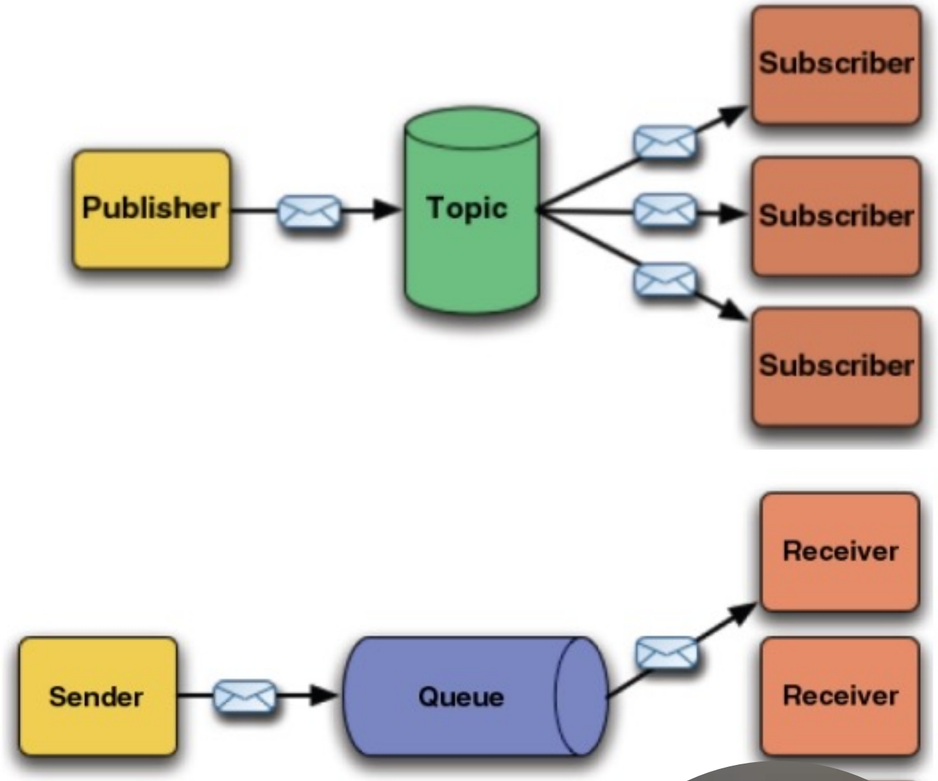
Events & Implicit invocation

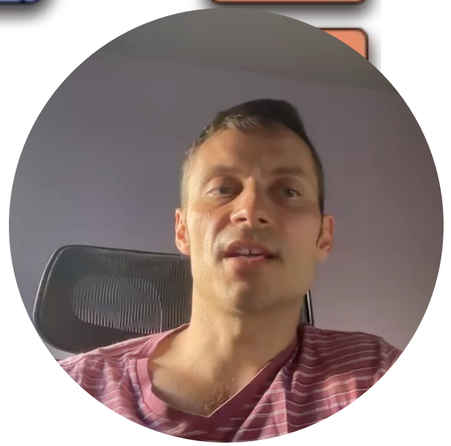
- Back to blackboard



Events & Implicit invocation

- Explicit vs. Implicit invocation
  - Events
    - Message broker





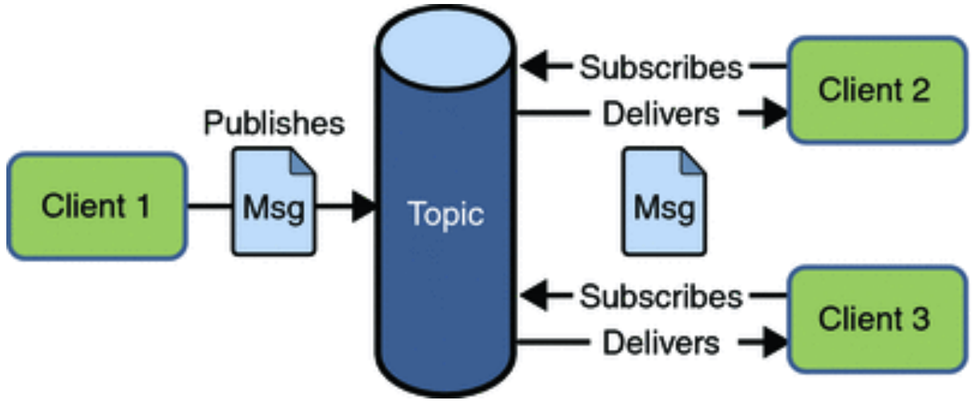
Events & Implicit invocation

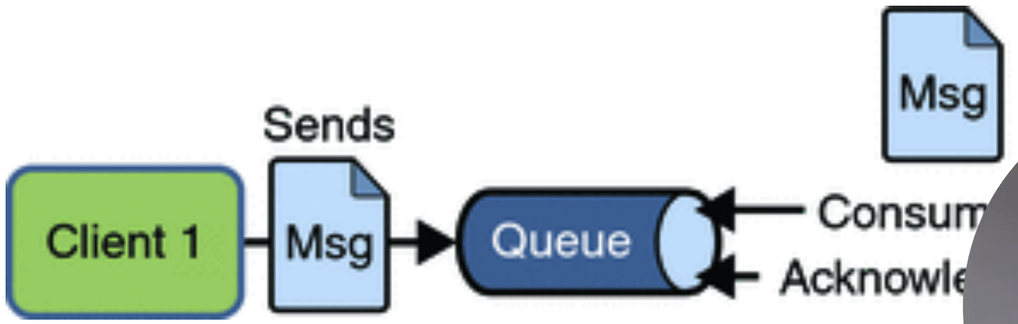
- +Nice problem decomposition
  - +Independence
- +System maintenance and reuse
- +Performance boost

- - We lose the holistic perspective
- - Hard to understand and debug
- - Performance

Events & Implicit invocation

- Explicit vs. Implicit invocation
  - Events
    - Message broker

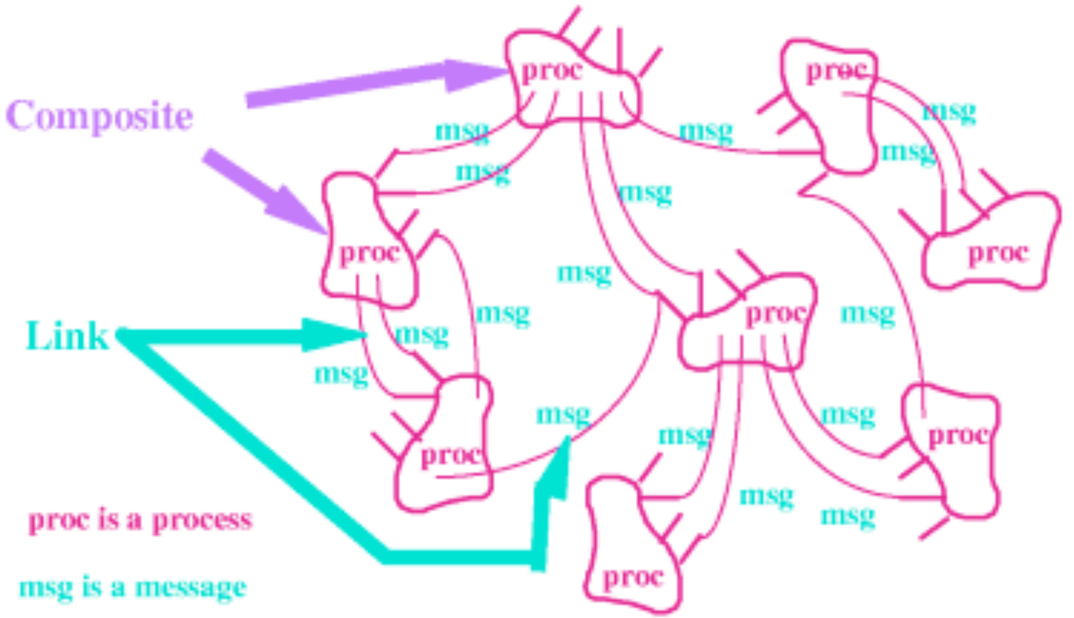




https://docs.oracle.com/

Independent components

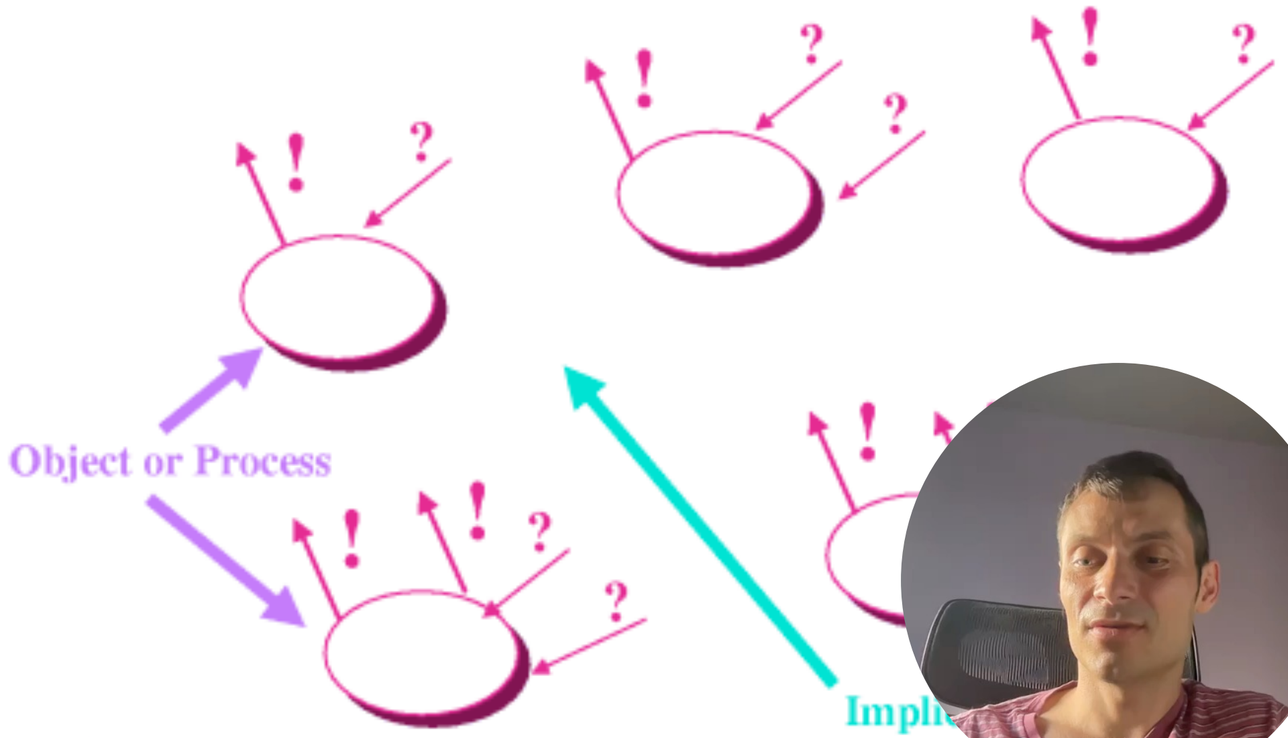
- Instead of objects in call and return we introduce independent processes
  - Decentralization
- Explicit invocation



http://www.cs.cmu.edu/

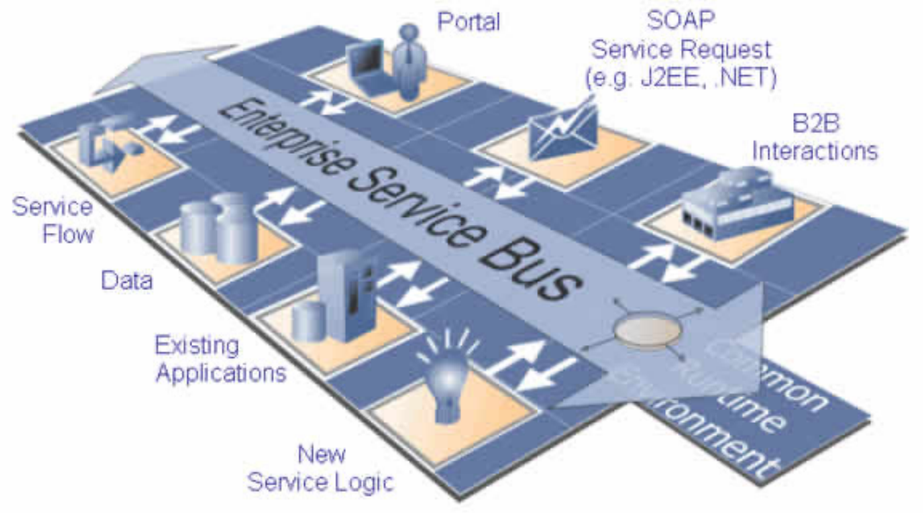
Independent components

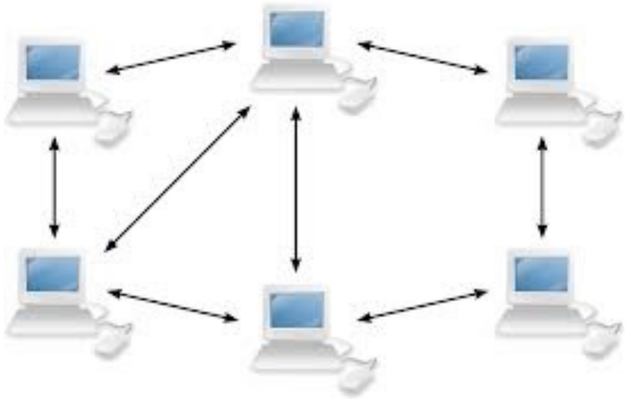
- Implicit invocation through independent components
  - Message manager

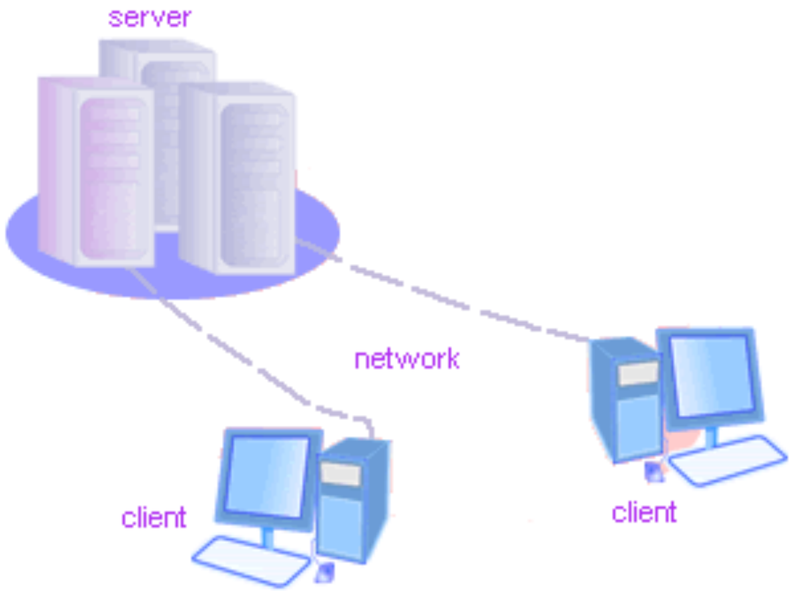


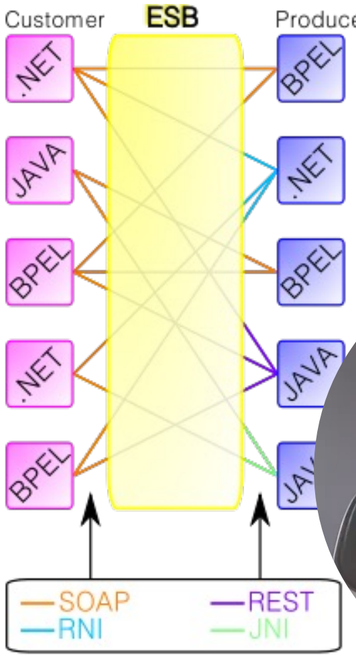
Independent components

- Client-server
- Peer to peer
- Enterprise service bus



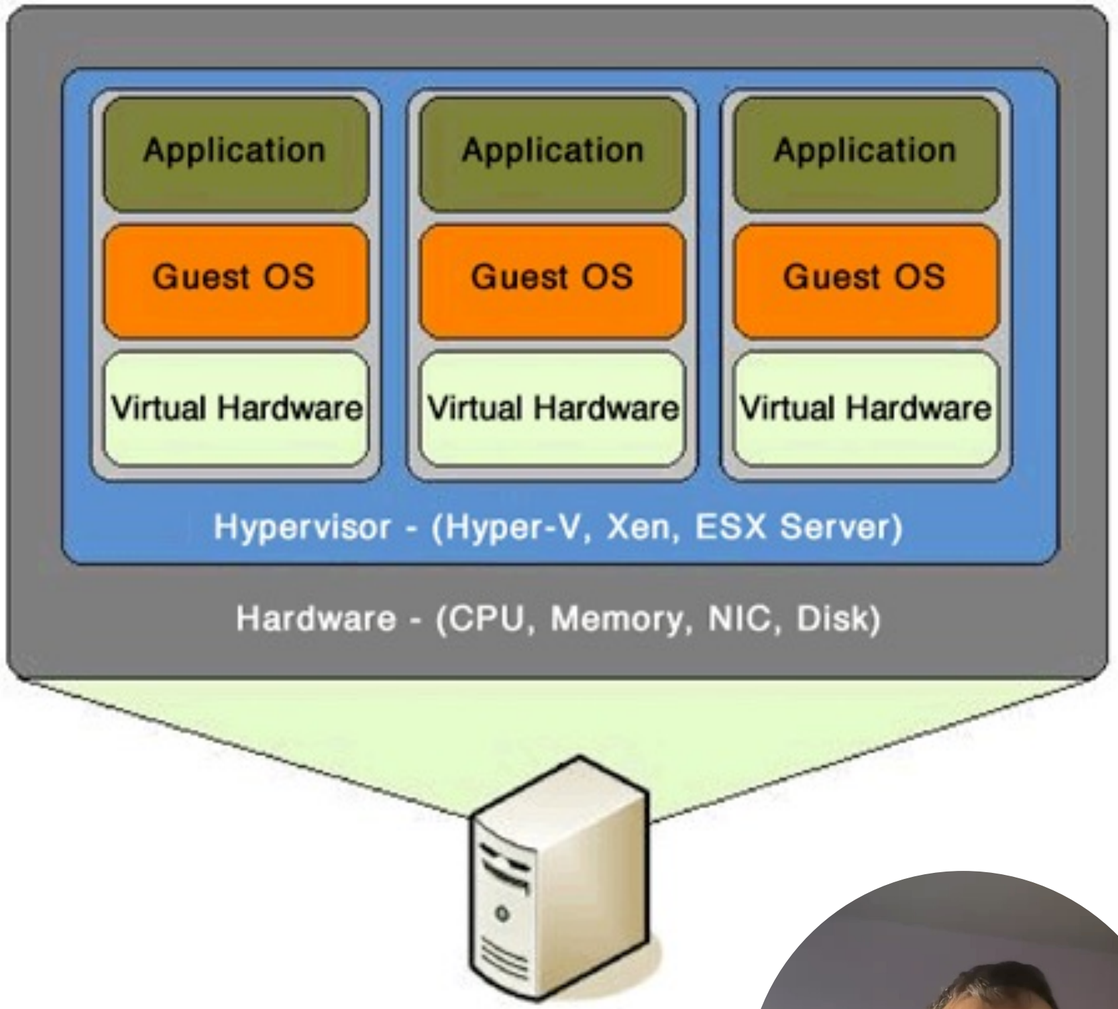




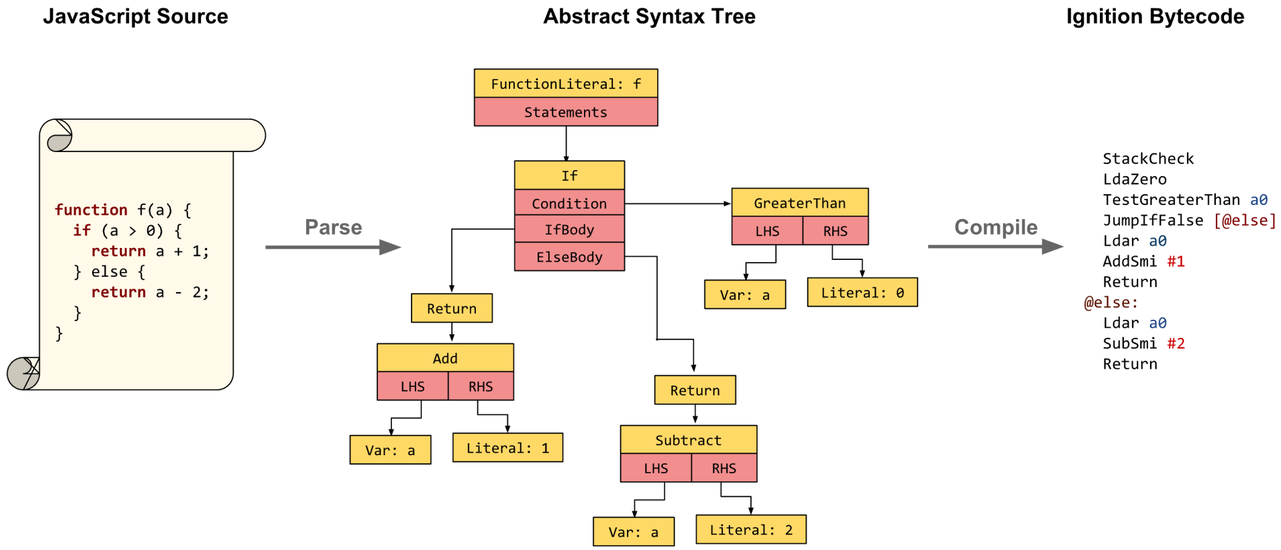


Virtual machines

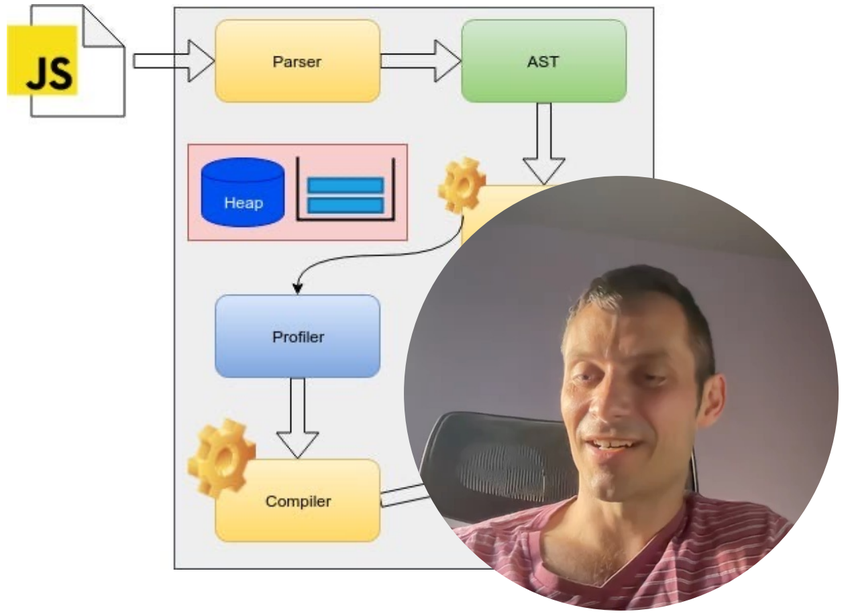
- Java Virtual Machine (JVM)

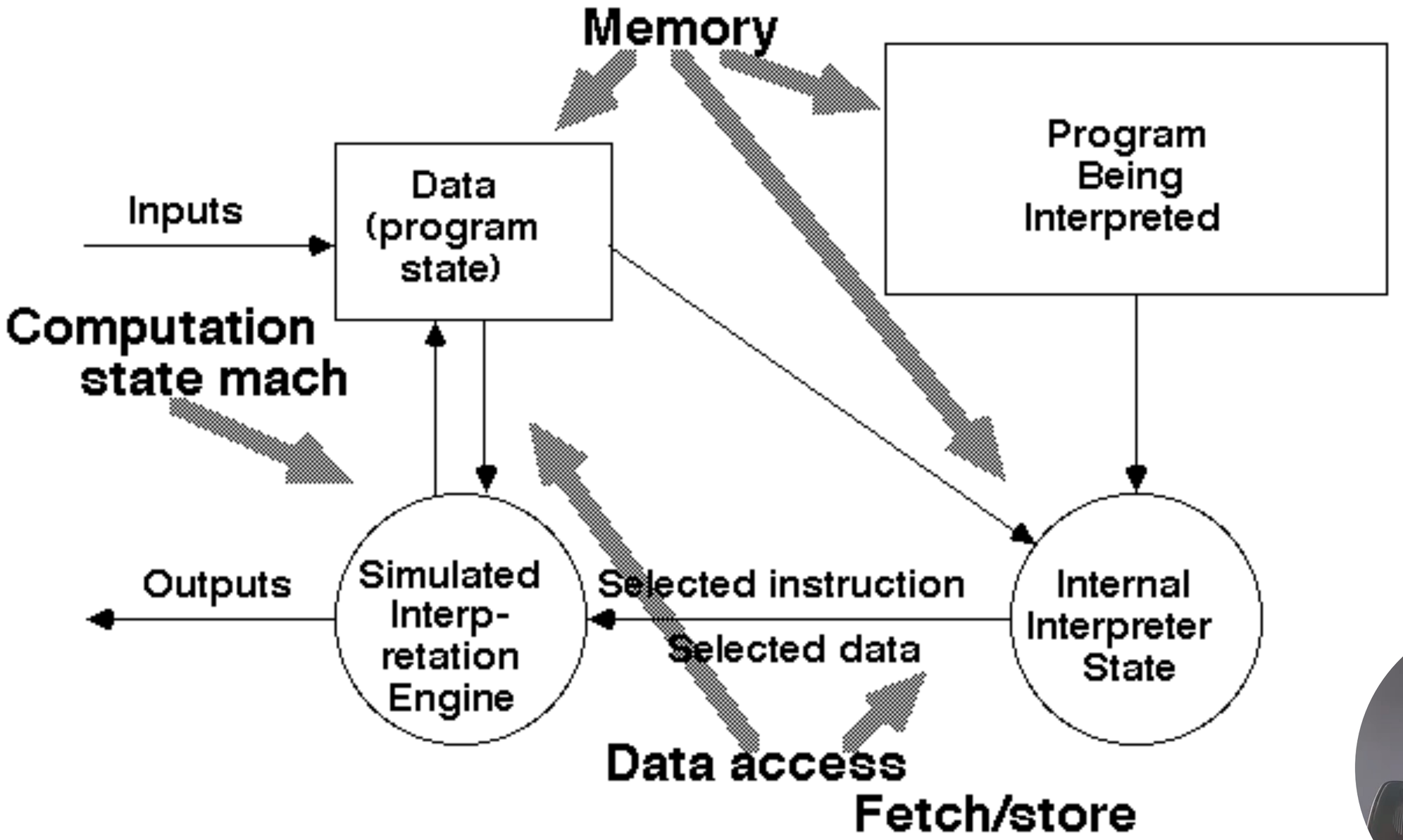


Virtual machines



- Interpreter
  - Flexible
    - Can interrupt at anytime and question or change the state
  - Performance impact
    - Indirection
  - Java Virtual Machine
  - PHP Interpreter
  - Java Script Interpreter




Memory
Program
Being
Interpreted
Inputs
Data
(program
state)
Computation
state mach
Outputs
Simulated
Interp-
retation
Engine
Selected instruction
Selected data
Internal
Interpreter
State
Data access
Fetch/store

Architecture Styles

- Data centric
  - Data integration, Distribution, Control, Coordination
  - Scalability, Low coupling, Centralization, Reuse, Modifiable,
- Data flow - Pipe and Filter
  - Modifiable, Reuse, Easy design, Simplicity, Low Coupling,
  - Slow, No filter cooperation, Lot of parsing
- Call and return
  - Modifiable, Reusable, Inf. hiding, Structural decomposition, Separation of concerns
- Layers – larger scale
- Implicit invocation
  - Modifiable, Low coupling, Hard to comprehend,
- Independent components
  - Integration, Scalability, Reuse, Low coupling, Distribution, Reliability
- Virtual Machines
  - Simulation, Emulation, Portability!, Flexibility, Low
    Performance, Extended fe

Organizing styles

- By component interaction
  - **Control**
  - **Data**
- More detail
  - How is it managed?
    - shared, allocated, transferred
  - What drives the data exchange?
  - How data and control corresponds?
- **Constituent parts**
  - Various types of **components** and **connectors**

Organizing styles – control

Coordination between components

1. **Topology**
    1. **Linear, cyclic, acyclic, tree, star, arbitrary**
    2. **One way, duplex**
    3. **Static, dynamic**
2. **Continuity of data**
    1. **Actions depend on states**
    2. **Synchronous/asynchronous**
3. **Binding time**
    1. **When do we transfer**
    2. **Determined at compile time/runtime/invocation time/write time**

Organizing styles – data

Coordination between components

1. **Topology**
    1. **Linear, cyclic, acyclic, tree, star, arbitrary**
    2. **One way, duplex**
    3. **Static, dynamic**
2. **Continuity**
    1. **Continuous/sporadic**
    2. **High/low volume**
3. **Mode – availability in the system**
    1. **Pasted from component to component – OOD**
    2. **Shared**
    3. **Copy out/copy in**
    4. **Broadcast/Multicast/Unicast**
4. **Bind time**
    1. **When to start transfer**

Organizing styles – control vs. data

- Coordination between components
  1. **Topology match**
     1. **Isomorphism**
  2. **Direction of control and data**
     1. **Same/opposite**
     2. **High/low volume**

Frameworks components/control/data

Style	Constituent Parts		Control Issues		Data Issues		Control/Data Interaction
	Components	*Connectors*	*Topology*	*Synchronicity*	*Topology*	*Continuity*	*Isomorphic Shapes*
Data Flow Architectural Styles							
Batch Sequential	Stand-alone programs	Batch data	Linear	Sequential	Linear	Sporadic	Yes
Data-Flow Network	Transducers	Data Stream	Arbitrary	Asynchronous	Arbitrary	Continuous	Yes
Pipes and Filter	Transducers	Data Stream	Linear	Asynchronous	Linear	Continuous	Yes
Call and Return							
Main Program/ Subroutines	Procedure	Procedure Calls	Hierarchical	Sequential	Arbitrary	Sporadic	No
Abstract Data Types	Managers	Static Calls	Arbitrary	Sequential	Arbitrary	Sporadic	Yes
Objects	Managers	Dynamic Calls	Arbitrary	Sequential	Arbitrary	Sporadic	Yes
Call based Client Server	Programs	Calls or RPC	Star	Synchronous	Star	Sporadic	Yes
Layered	-	-	Hierarchical	Any	Hierarchical	Sporadic	Often
Independent Components							
Event Systems	Processes	Signals	Arbitrary	Asynchronous	Arbitrary	Spor	
Communicating Processes	Processes	Message Protocols	Arbitrary	Any but Sequential	Arbitrary	Spo	
Data Centered							
Repository	Memory, Computations	Queries	Star	Asynchronous	Star	Spo	
Blackboard	Memory, Components	Direct Access	Star	Asynchronous	Star	Sporadi	

Frameworks
components/control/data

Table 1: Specializations of the dataflow network style

Style	Constituent parts		Control issues			Data issues				Ctrl/data interaction	
	Comp-onents	Conn-ectors	Topo-logy	Synch-ronicity	Bind-ing time	Topo-logy	Contin-uity	Mode	Bind-ing time	Isomor-phic shapes	Flow dir-ections
Data flow styles: Styles dominated by motion of data through the system, with no "upstream" content control by recipient											
Dataflow network [B+88]	trans-ducers	data stream	arbi-trary	asynch	i, r	arbi-trary	cont lvol or hvol	passed	i, r	yes	same
• Acyclic [A+95]			acyclic			acyclic					
• Fanout [A+95]			hier-archy			hier-archy					
• Pipeline [DG90, Se88, A+95]			linear			linear					
-Unix pipes and filters [Ba86a]		ascii stream			i						
Key to column entries											
Synchronicity	asynch (asynchronous)										
Binding time	i (invocation-time), r (run-time)										
Continuity	cont (continuous), hvol (high-volume), lvol (low-volume)										

Frameworks
components/control/data

Table 2: Specializations of the interacting processes style

Style	Constituent parts		Control issues			Data issues				Ctrl/data interaction	
	Components	Connectors	Topology	Synchronicity	Binding time	Topology	Continuity	Mode	Binding time	Isomorphic shapes	Flow directions
Interacting process styles: Styles dominated by communication patterns among independent, usually concurrent, processes											
Communicating processes [An91, Pa85]	processes	message protocols	arb	any but seq	w, c, r	arb	spor lvol	any	w, c, r	possibly	if isomorphic either
One-way data flow, networks of filters			linear	asynch		linear		passed	w, c	yes	same
Client/server request/reply			star	synch		star		passed		yes	opposite
Heartbeat			hier	ls/par		hier or star		passed shared ci/co		no	same
Probe/echo			incomplete graph	asynch		incomplete graph		passed		yes	same
Broadcast			arb	asynch		star		bdcast		no	same
Token passing			arb	asynch		arb.		passed			
Decentralized servers											
Replicated workers			hier	synch		hier		passed shared			
Key to column entries											
Topology	hier (hierarchical), arb (arbitrary), star, linear (one-way)										
Synchronicity	seq (sequential, one thread of control), ls/par (lockstep parallel), synch (synchronous), opp (opportunistic)										
Binding time	w (write-time--that is, in source code), c (compile-time), i (invocation-time), r (ru										
Continuity	spor (sporadic), lvol (low-volume)										
Mode	shared, passed, bdcast (broadcast), mcast (multicast), ci/co (copy-in/copy-out)										

Case Study

Key Word in Context = KWIC

Key Word in Context

- Indexing system
- Input:
  - Text lines
  - Line contains words
- Output:
  - List of line circular shifts in alphabetical order

Key Word in Context

- Input:

Department of Computer Science
Key Word in Context

- Output:

Computer Science Department of
Context Key Word in
Department of Computer Science
Key Word in Context
Science of Computer Science
Word in Context Key
in Context Key Word
of Computer Science Department

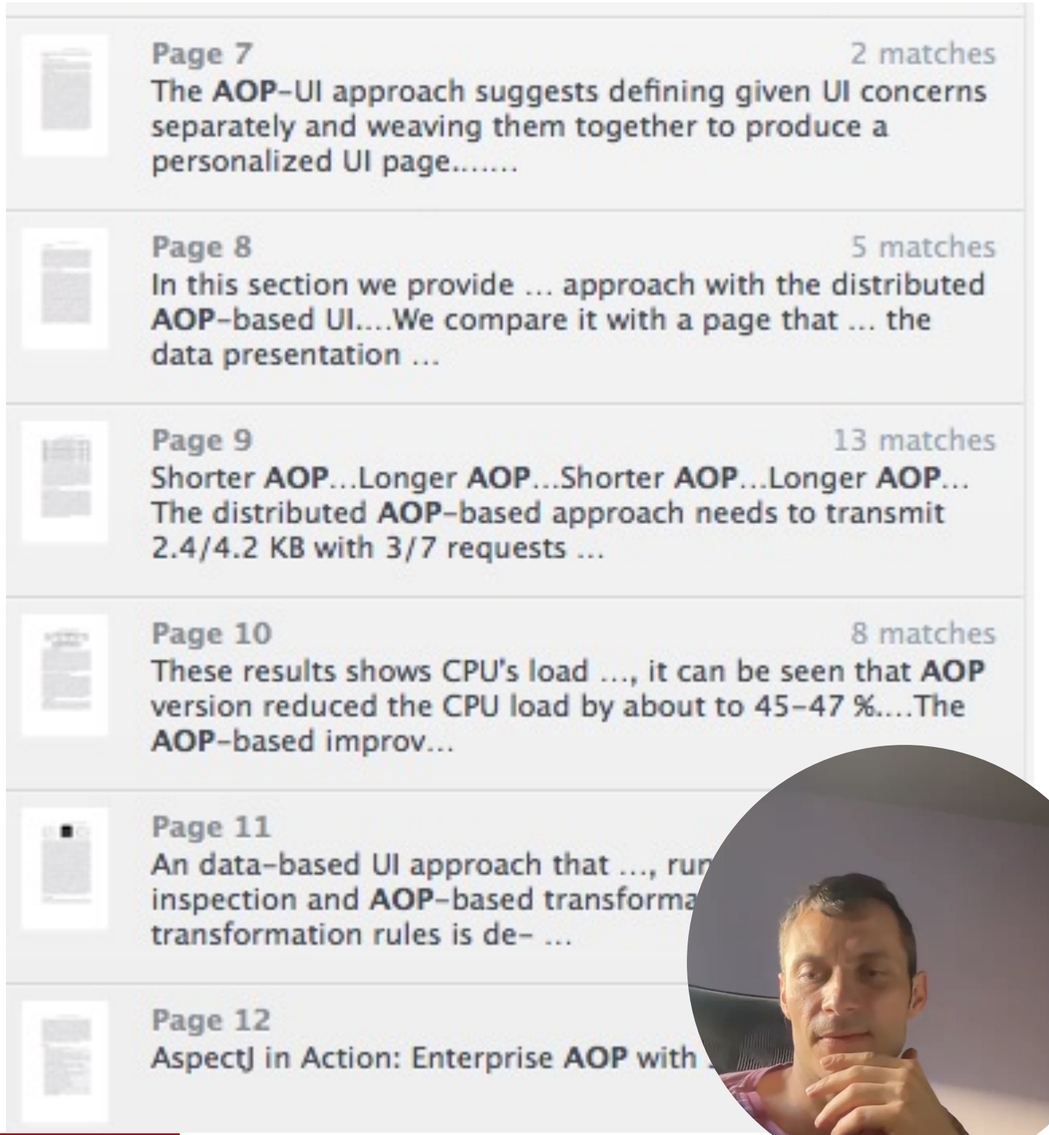
Department of >>Computer<< Science
Key Word in >>Context<<
>>Department<< of Computer Science
>>Key<< Word in Context
Department of Computer >>Science<<
Key >>Word<< in Context
Key Word >>in<< Context
Department >>of<< Computer Scie

Key Word in Context – the story behind

KWIC is an **acronym** for Key Word In Context, ...	page 1
... Key Word In Context, the most **common** format for concordance lines.	page 1
... the most common format for **concordance** lines.	page 1
... is an acronym for Key Word In **Context**, the most common format ...	page 1
Wikipedia, The Free **Encyclopedia**	page 0
... In Context, the most common **format** for concordance lines.	page 1
Wikipedia, The **Free** Encyclopedia	page 0
KWIC is an acronym for **Key** Word In Context, the most ...	page 1
KWIC is an acronym for Key Word ...	page 1
... common format for concordance **lines**.	page 1
... for Key Word In Context, the **most** common format for concordance ...	page 1
Wikipedia, The Free Encyclopedia	page 0
KWIC is an acronym for Key **Word** In Context, the most common ...	page 1

Key Word in Context

MAC Preview

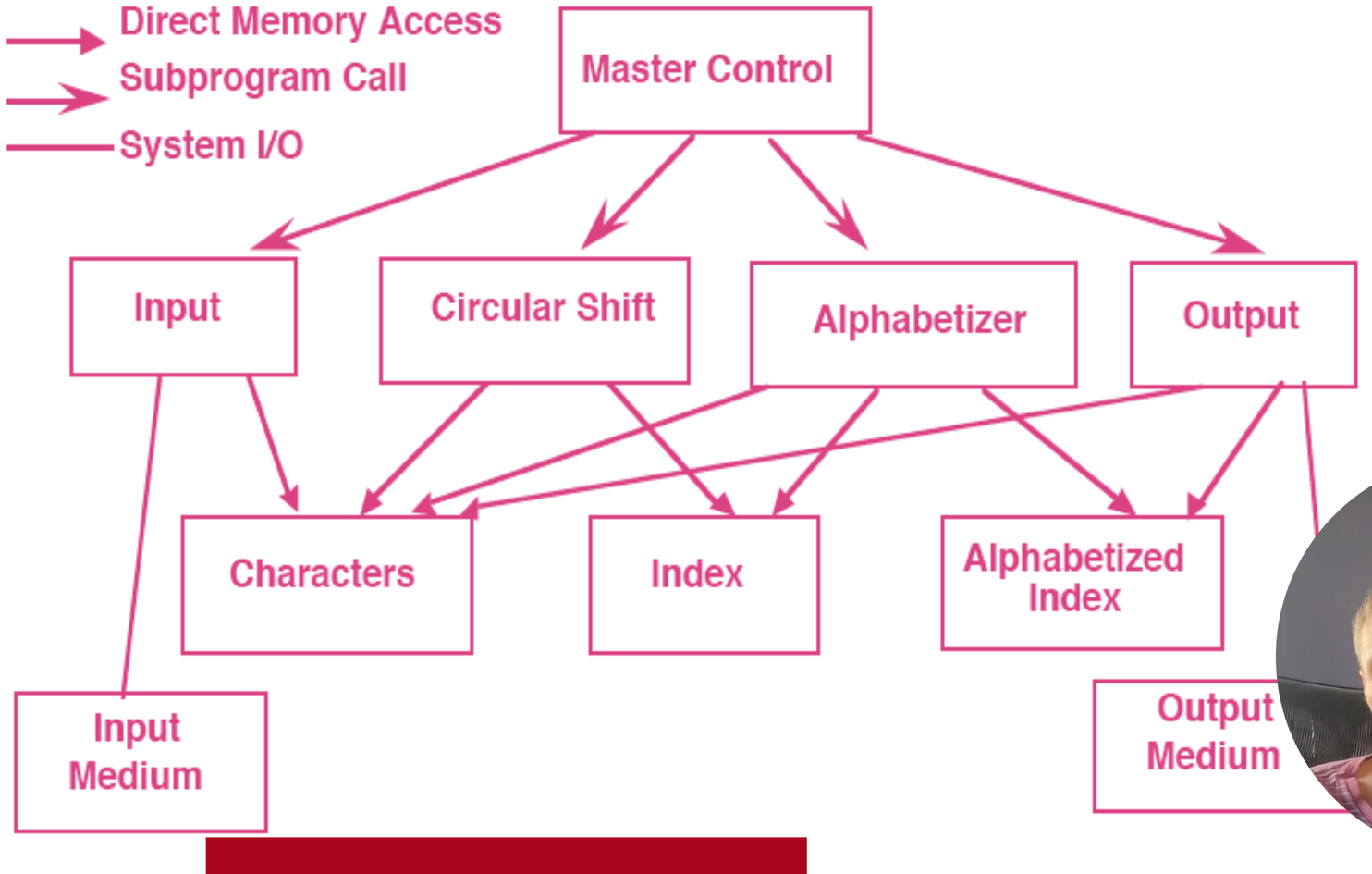


Key Word in Context

Styles:

- Main procedure with subprocedures with shared memory
  - No objects, no collections
- Object-oriented
- Event-based system
- Pipes and filters

Main procedure with subprocedures with shared memory



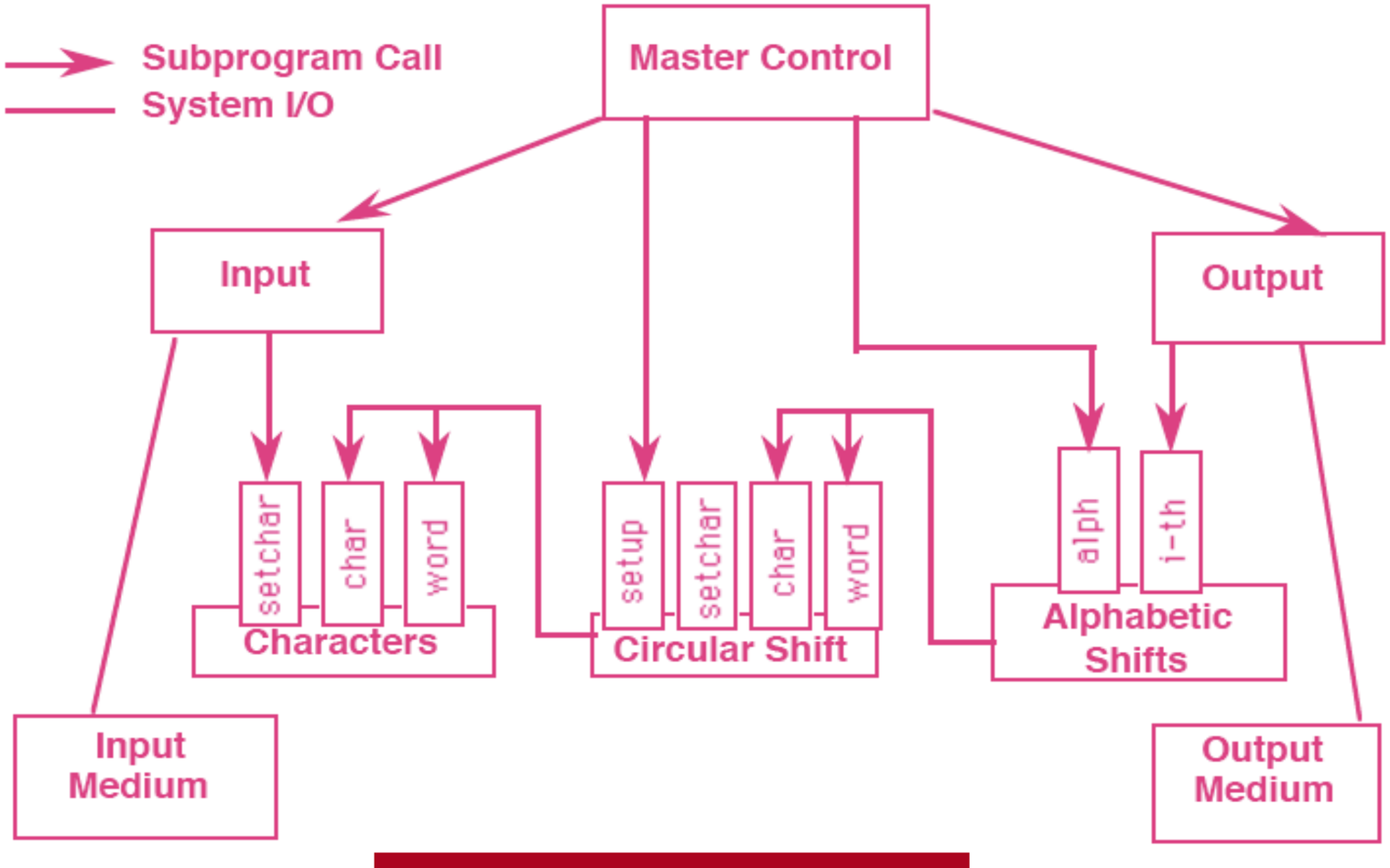
Main procedure with subprocedures with shared memory

```
public class KWIC {
  private char[] chars_;
  private char[] shift_chars_;
  private int[] line_index_;
..
  public void input(String file) {..}
  public void circularShift() {..}
  public void alphabetizing() {..}
  public void output() {..}
```

Main procedure with subprocedures with shared memory

```
..
  public static void main(String[] args) {
   KWIC kwic = new KWIC();
   if (args.length != 1) {
      System.err.println("specify filename");
      System.exit(1);
   }
   kwic.input(args[0]);
   kwic.circularShift();
   kwic.alphabetizing();
   kwic.output();
  }
```

Object-oriented



Object-oriented - classes

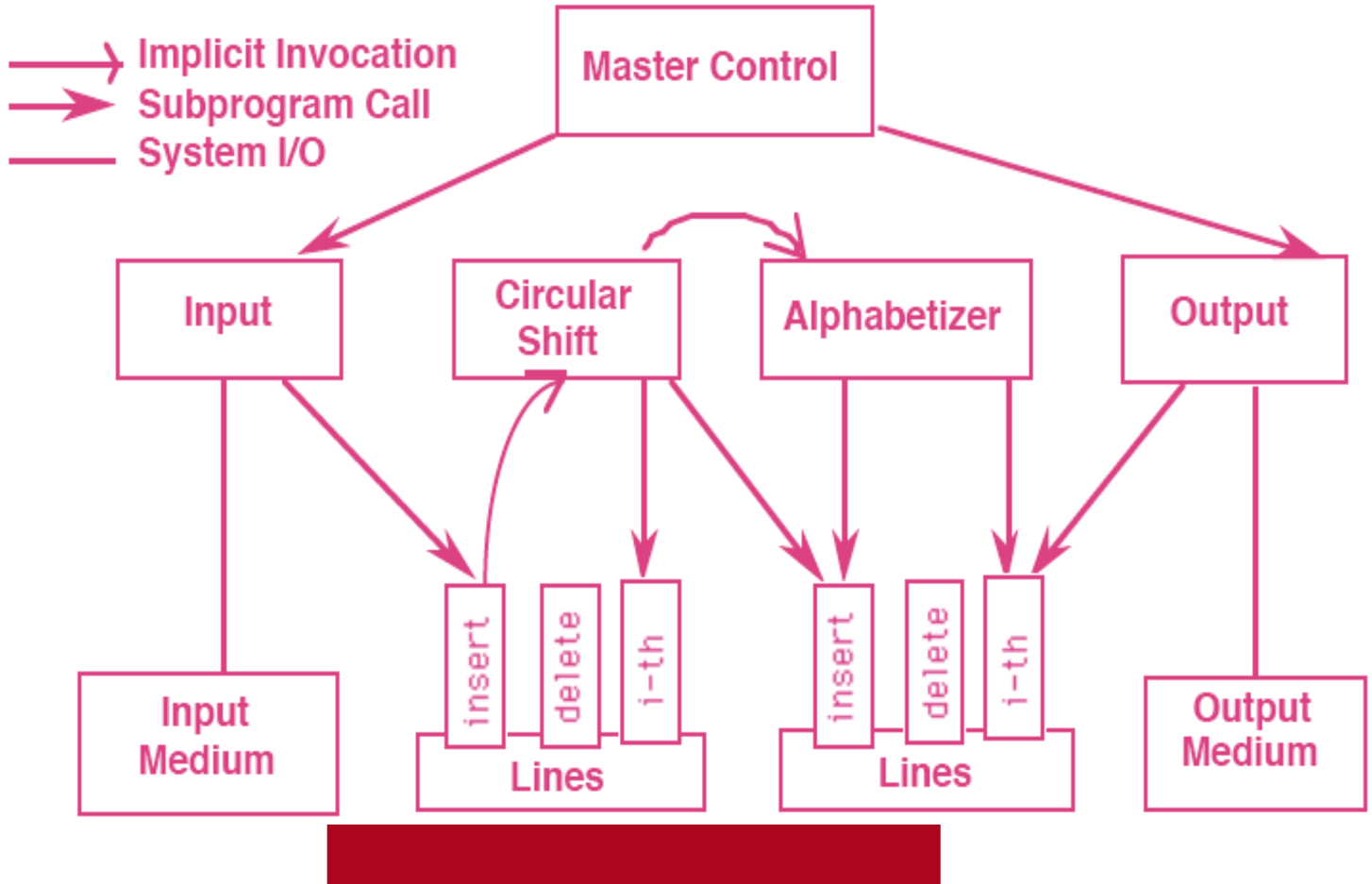
- Alphabetizer.java
- CircularShifter.java
- Input.java
- KWIC.java
- LineStorage.java
- Output.java

Object-oriented - classes

```
public void execute(String file){
  LineStorage lines = new LineStorage();
  Input input = new Input();
  CircularShifter shifter = new CircularShifter();
  Alphabetizer alphabetizer = new Alphabetizer();
  Output output = new Output();

  input.parse(file, lines);
  shifter.setup(lines);
  alphabetizer.alpha(shifter);
  output.print(alphabetizer);
}
```

Event-based


Implicit Invocation
Subprogram Call
System I/O
Master Control
Input
Circular Shift
Alphabetizer
Output
Input Medium
insert
delete
i-th
Lines
insert
delete
i-th
Lines
Output Medium

Event-based

```
public class CircularShifter
                    implements Observer {
..
public class Alphabetizer
                    implements Observer {
..
public class LineStorageWrapper
                    extends Observable{
```

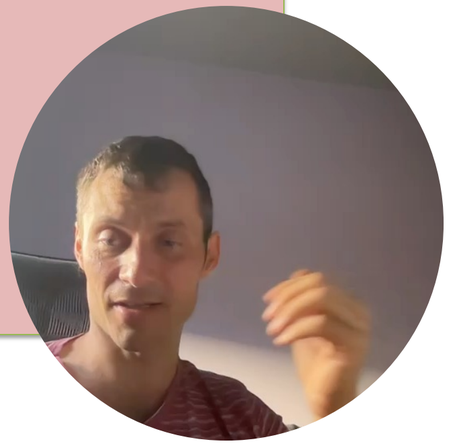
Event-based

```
public class LineStorageWrapper extends Observable{
    ..
  public synchronized void addObserver(Observer o) {
      if (!obs.contains(o)) {
              obs.addElement(o);
          }
      }
..
   public void addLine(String[] words){
          lines_.addLine(words);
          LineEvent event = new LineEvent(Event.ADD);
          setChanged();
          notifyObservers(event);
      }
```

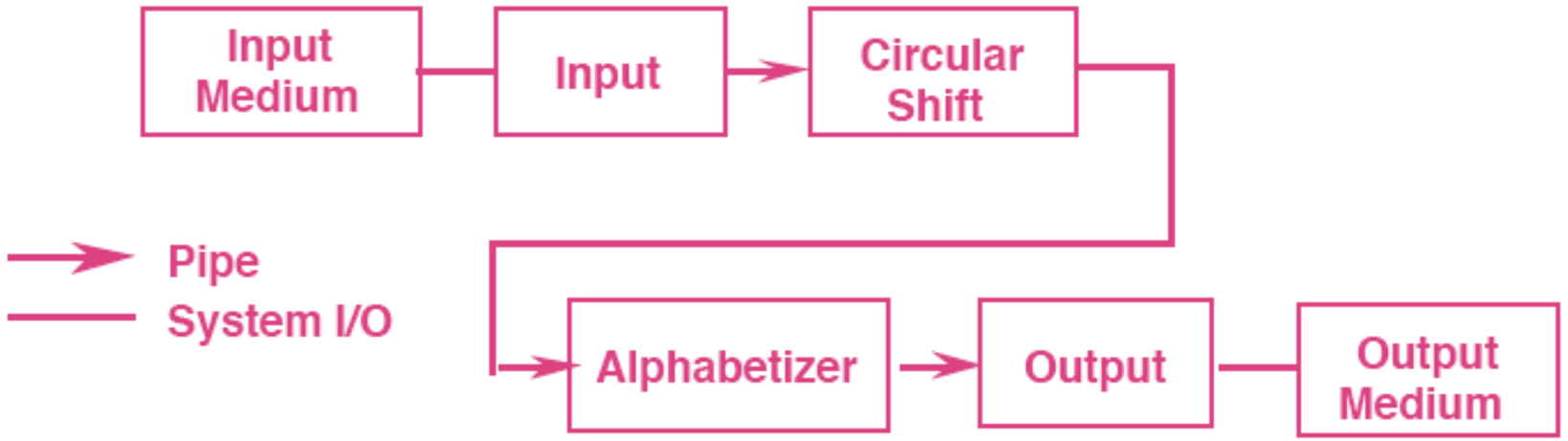
Event-based

```
public void execute(String file) {
   LineStorageWrapper lines = new LineStorageWrapper();
   LineStorageWrapper shifts = new LineStorageWrapper();

   Input input = new Input();
   CircularShifter shifter = new CircularShifter(shifts);
   lines.addObserver(shifter);
   Alphabetizer alphabetizer = new Alphabetizer();
   shifts.addObserver(alphabetizer);
   Output output = new Output();
   input.parse(file, lines);
   output.print(shifts);
}
```



Pipes and Filters



Pipes and Filters - classes

- Alphabetizer.java
- CircularShifter.java
- **Filter.java**
- Input.java
- KWIC.java
- Output.java
- **Pipe.java**

Pipes and Filters

```
public class Filter implements Runnable{
   protected Pipe input;
   protected Pipe output;
..
public class Pipe {
   private PipedReader reader;
   private PipedWriter writer;
```

Pipes and Filters

```
public void execute(String file){
   Pipe inCS = new Pipe();
    Pipe csAl = new Pipe();
    Pipe alOu = new Pipe();

    FileInputStream in = new FileInputStream(file);
    Input input = new Input(in, inCS);
    CircularShifter shifter =new CircularShifter(inCS,csAl);
    Alphabetizer alpha = new Alphabetizer(csAl, alOu );
    Output output = new Output(alOu );

    input.start();shifter.start();
    alpha.start();output.start();
}
```