Architektury Softwarových Systémů

Ing. Tomáš Černý M.S.C.S.

Prozatím odkazy www

..

http://groups.google.com/group/36ass-course-14

Zaregistrujte se!

Organizace

Tomáš Černý

Karlovo náměstí 4. patro #333

email: tomas.cerny@fel.cvut.cz

diskuze: http://groups.google.com/group/36ass-course-14

tel: Sony

Akademická čest

Akademická čest a etika budiž brána vážně. Mnoho studií říká, že někteří studenti podvádí z ignorantství, nejistoty, a nejasností jaké chování je nečestné.

Sheilah Maramark, Mindi Barth Maline

- U.S. Dept. of Education, Office of Research, August 1993

Co je to podvod?

- Použití zdroje jiného než doporučené literatury, přednášek či cvičení pro vaše úkoly, projekty a zkoušky
- Kopie cizí práce, i nahlédnuti je kopírování

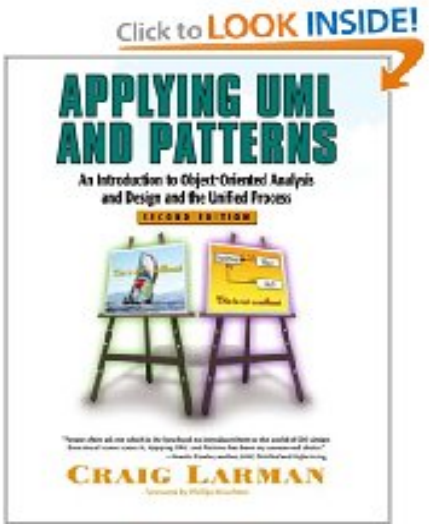
Akademická čest

- Poskytnutí vaší práce někomu jinému
- Spolupráce s další osobou aniž by bylo zadáno.
  - Diskuze s další osobou o úkolu, projektu, písemce..
- Používání poznámek, knih při zkoušce
- Poskytnutí odpovědí dalším studentům
- Náhled to ukradené kopie testu
- Plagiátorství
- Studie zkoušek z loňského roku
- Poskytnutí testu či úkolů z loňského semestru
- Zkouška za někoho dalšího
- Předložení cizí práce
- Poskytnutí otázek studentům z jiné skupiny
- Studie minulých testů instrutora bez povolení

(Ref: Greg Hamerly, Baylor, TX - http://cs.ecs.baylor.edu/~hamerly/courses/3334_09f/)

Knihy

- Craig Larman



- http://www.pdf-search-engine.com/larman-uml-pdf.html
- http://books.google.cz/books?id=r8i-4En_aa4C&printsec=frontcover&dq=applaying+uml+and+patterns&cd=1#v=onepage&q=&f=false

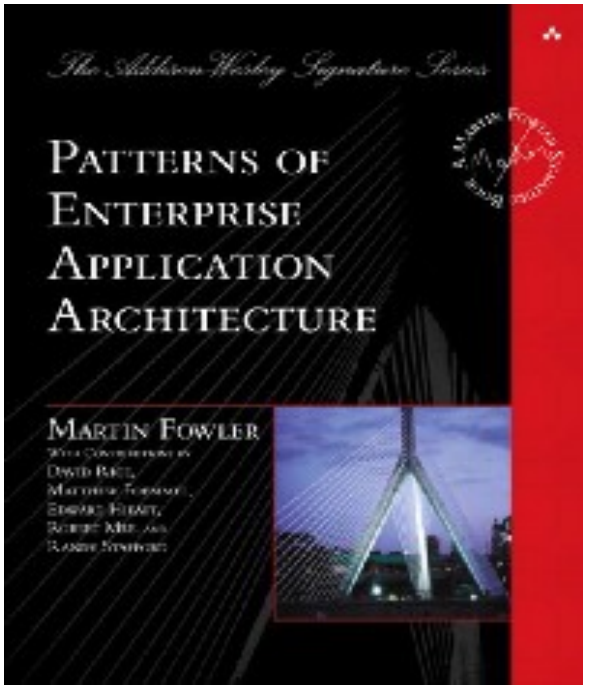
Knihy

- GoF

- Erich Gamma

Knihy

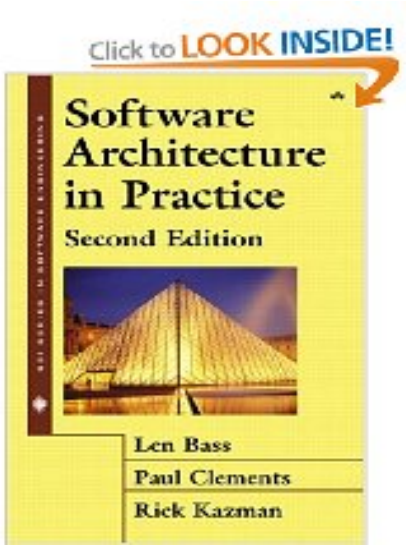
- PEAA



- Fowler
  - http://martinfowler.com/eaaCatalog/
  - http://books.google.cz/books?id=FyWZt5DdvFkC&lpg=PA1&dq=Patterns%20of%20Enterprise%20Application%20Architecture&pg=PT26#v=onepage&q=&f=false

Knihy

- SAP



- Bass
  - http://books.google.cz/books?id=mdiIu8Kk1WMC&lpg=PP1&dq=software%20architectur%20in%20practice&pg=PR4#v=onepage&q=&f=false

Úvod – organizace předmětu

- Tým po dvou lidech
  - Mimo testy a články
  - Doporučuji najít někoho koho znáte
- Hodnocení
  - Student a tým získává iDollary za odevzdanou práci
    - Tyto dolary se ukládají do ČeBanky
    - Každých 20 dní je dedukováno $10
    - Nutný je pozitivní balanc
      - Možná jedna půjčka na 20 dní, úrok je 100%
  - Nedostatek financí => vyhladovění a eliminace

Úvod – za co mohu získat iDollary?

- Přednášky
  - Prezentace vzoru (architektonický/eaa) **($10)**
    - Viz slidy GOF + nutná přidaná hodnota a otestovaný/naprogramovaný vzor
  - LaTex článek GOF **($10)**
  - Test **($20)**
- Cvičení
  - KWIC **(4 x implementace po $3 + benchmark $5)**
  - Vědecké Články **(+/- $)**
  - Projekt – prezentovaný vzor + 3 další **($10)**
  - Test **($20)**
- Cvičení/Challenge
  - Každých 14 dní challenge! **(Extra $)**

Úvod

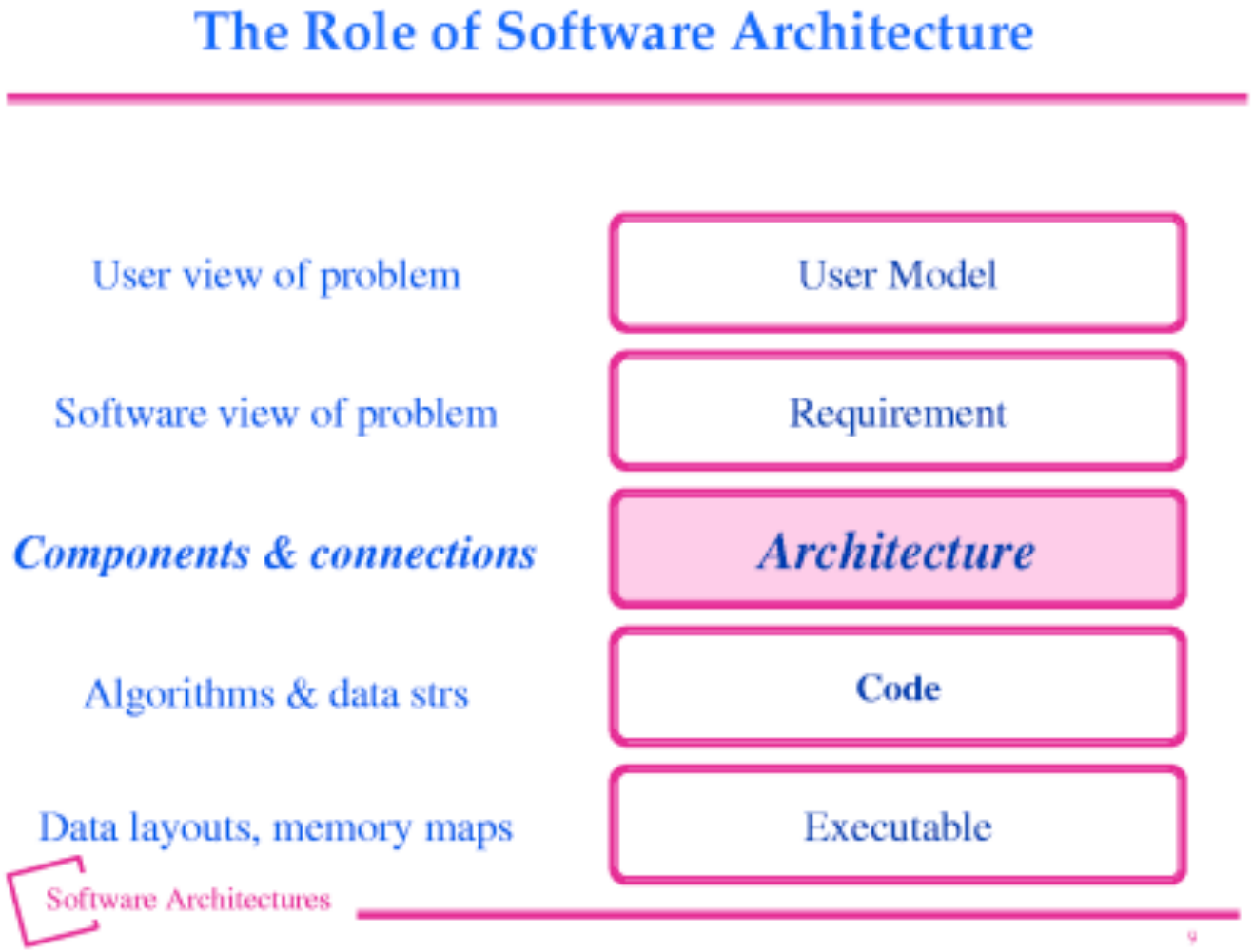
- Hodnocení
  - 3-4 nejlepší A
  - 3-4 další nejlepší B
  - ..
  - Až po F

  - Tým s maximem iDollarů z challenge má automatické A pokud splní limity a poviné úkoly

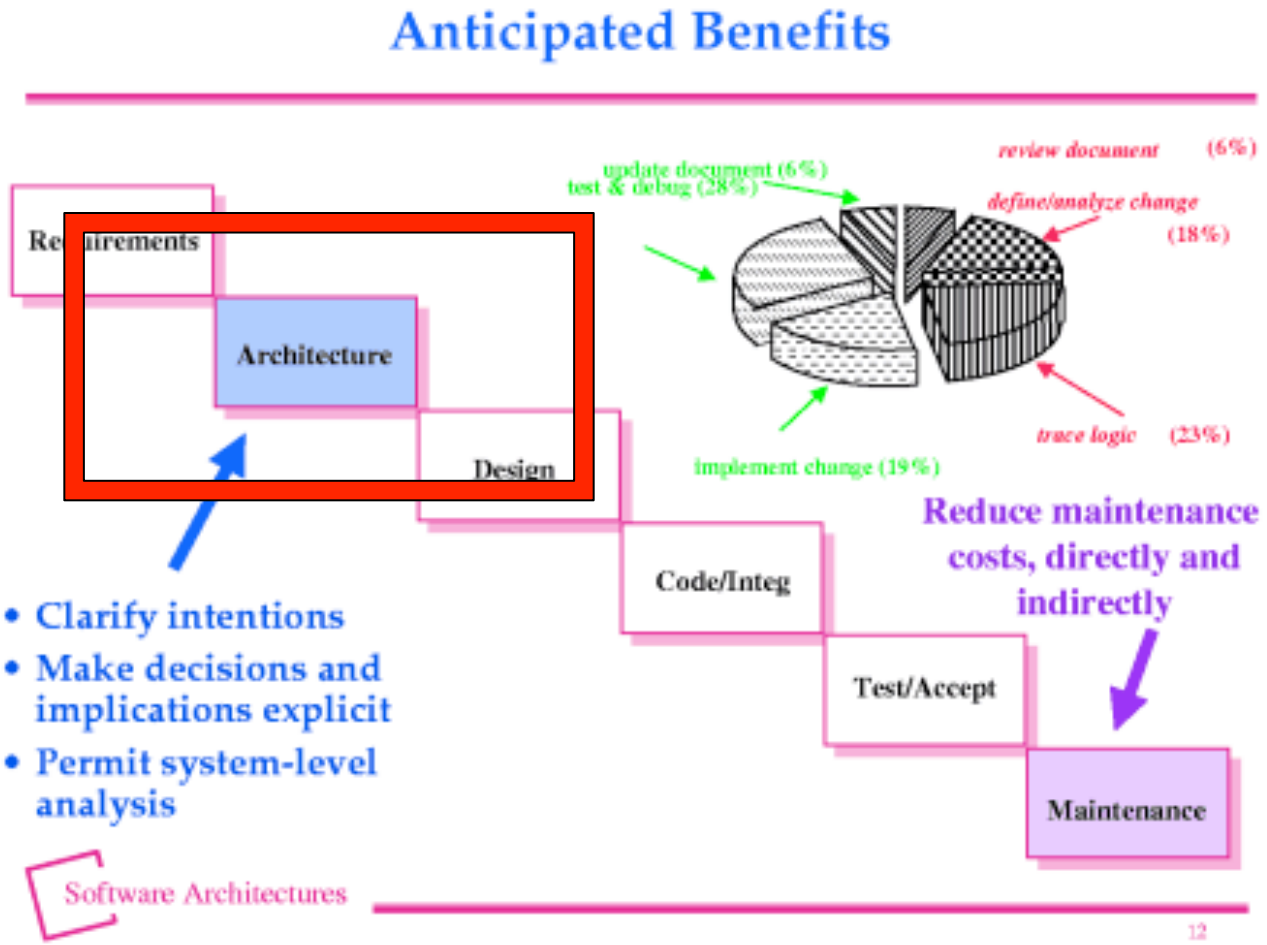
Obsah

- Architektonické styly

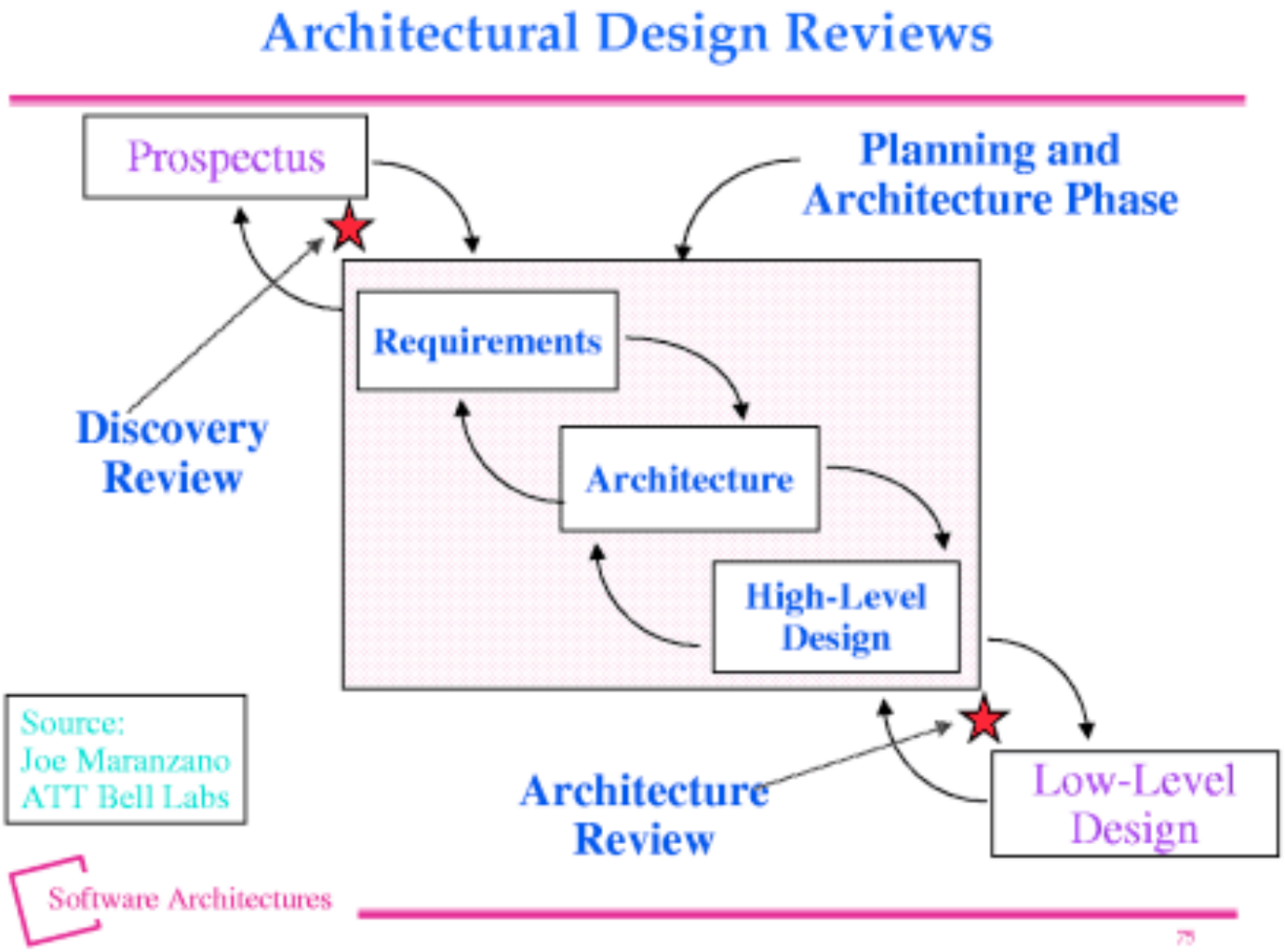
Intro


The Role of Software Architecture
User view of problem
User Model
Software view of problem
Requirement
Components & connections
Architecture
Algorithms & data strs
Code
Data layouts, memory maps
Executable
Software Architectures
9

Intro


Anticipated Benefits
update document (6%)
test & debug (28%)
review document (6%)
define/analyze change (18%)
trace logic (23%)
implement change (19%)
Re irements
Architecture
Design
Code/Integ
Test/Accept
Maintenance
Reduce maintenance costs, directly and indirectly
• Clarify intentions
• Make decisions and implications explicit
• Permit system-level analysis
Software Architectures
12

Intro


Architectural Design Reviews
Prospectus
Planning and Architecture Phase
Requirements
Discovery Review
Architecture
High-Level Design
Source: Joe Maranzano ATT Bell Labs
Architecture Review
Low-Level Design
Software Architectures
75

Intro

Architectural Design Task

Different issues for architecture & programs

Architecture	*Programs*
interactions among parts	implementations of parts
structural properties	computational properties
declarative	operational
mostly static	mostly dynamic
system-level performance	algorithmic performance
outside module boundary	inside module boundary
composition of subsystems	copy code or call libraries

Styl, referenční model a ref. architektura

Ve smyslu architektury definujeme následující pojmy

1. Styl architektury
2. Referenční model
3. Referenční architektura

Styl, referenční model a ref. architektura

1. Styl architektury

- Popis:
  - typů komponent,
  - vzorů pro kontrolu jejich běhu
  - a datových přenosů.
- Chápán jako množina omezení.
  - tato omezení definují rodinu architektur, které je umožňují / splňují.

Styl, referenční model a ref. architektura

2. Referenční model

Dělení funkcionality s datovým tokem mezi bloky.

Rozdělení problému (funkčnosti, datových toků)

- nedoporučuje konkrétní softwarové řešení

Je charakteristikou povahy domény.

Příklad:

- Zkuste vyjmenovat std. části kompileru?
- Či databáze?

Styl, referenční model a ref. architektura

3. **Referenční architektura**

Referenční model mapovaný na SW komponenty.

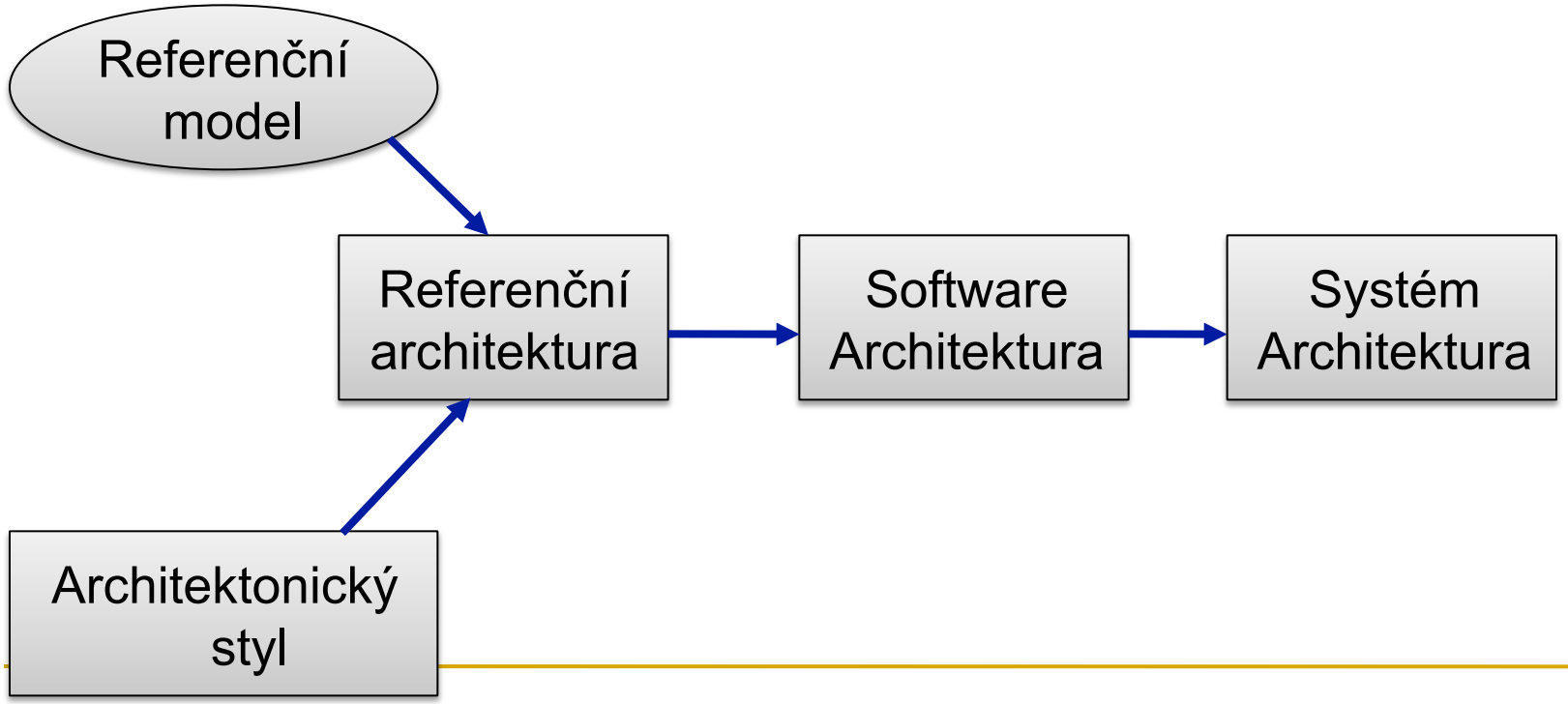
A na datový tok mezi komponentami.

Ref. model rozděluje funkcionalitu
Ref. Architektura = mapování této funkcionality na dekompozici systému

Komponenta může implementovat část funkce ale i mnoho funkcí.

Shrnutí

- Referenční model
- Architektonické styly
- Referenční architektury
  - nejsou architektury, ale užitečné mezikroky.



Definice softwarové architektury

"Architektura softwarového systému představuje definici struktury systému. Tuto strukturu tvoří softwarové **komponenty**, ze kterých se systém skládá, z vnějšku viditelné **vlastnosti** těchto komponent a **vztahy** mezi nimi."

Bass, Clements, Kazman. Software Architecture in Practice, Addison-Wesley 1997

Definice softwarové architektury

“Architektura je organizační struktura systému. Architektura může být rekurzivní dekomponována na části, které komunikují prostřednictvím rozhraní, na vztahy propojující jednotlivé části a omezení, která na jsou na jednotlivé části kladena. Komunikující složky zahrnují třídy, komponenty a subsystémy. ”

UML 1.3

Motivace

- Velikost a komplexita SW roste
  - Algoritmy a datové struktury jsou „malé“ stavební bloky
- Strukturní organizace
- Globální řízení
- Protokoly a synchronizace
- Funkcionalita
- Distribuce ?
- Škálovatelnost a výkon !

Jak na to?

- Atributy kvality jsou obecný návod pro stavbu systému
  - Příliš neurčité, abstraktní
  - *Více o kvalitách v jiné přednášce*
- Architektonické vzory/styly
  - Dosažení arch. aspektů softwarové kvality
  - Vyskytují se prakticky ve všech částech vývoje
- My budeme rozlišovat dva typy
  - Architektonické styly
  - Návrhové vzory
    - GOF, EAA, Arch

Vzory

- Co mají vzory společného?
  - Předdefinované bloky,
  - Lze upravit, aby seděly pro náš kontext.
- Charakteristiky jsou známy a popsány
- Každý vzor reprezentuje
  - Návrhová rozhodnutí, ta již použita a otestována
  - Best Practice! (jsou i anti-vzory)
- Rozdíl - měřítko a doba vývoje, kdy se aplikují.

Architektonické styly

- Analogicky jako v budově
  - Gotika
  - Řecký styl
  - Renesance
  - Baroko

- Klíčové charakteristiky a zákony pro kombinování funkčnosti se zachováním integrity ale nic konkrétního.. Jen rysy..

Architektonické styly - popis

- Architektonický styl určen dle frameworku:
  1. Množina **komponent**, které mají danou funkci
     - *Data repository*
     - *Process*
     - *Procedura*
  2. **Topologická struktura** komponent indikuje run-time závislosti

Architektonické styly - popis

- Architektonický styl určen dle frameworku:
  3. Množiny **sémantických omezení**
     - *Repository* nesmí měnit své hodnoty
  4. Množiny **konektorů**
     - Řídí komunikaci, koordinaci, či kooperaci komponent
       - *Volaní rutin*
       - *Remote procedure call*
       - *Data stream*
       - *Event broadcast*
       - *Socket*

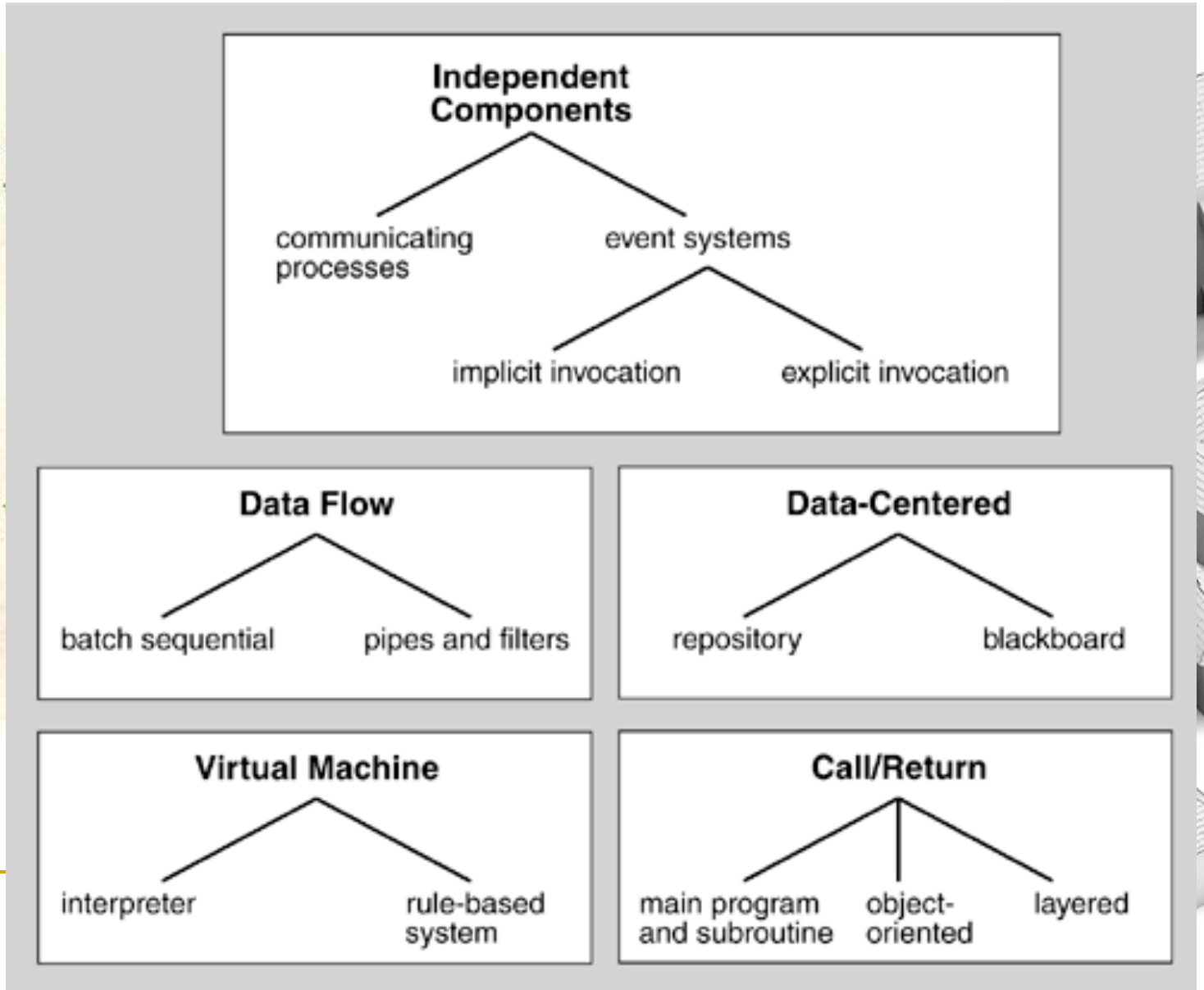
Architektonické styly

- **Styl není architektura**
  - Jako Gotika neurčuje jak přesně vypadá budova.
- **Styl definuje třídy architektur**
  - Abstrakce pro množinu architektur, které ji splňují
- Styly jsou často **nejednoznačné** z hlediska
  - Počtu zainteresovaných komponent
    - *Pipe & Filter* může mít dva filtry nebo deset
  - Interakce komponent
  - Funkce systému
    - Komponenta databáze, ale data se mění

Architektonické styly

- Styly se opakují systémovém návrhu
  - Ale i v různých formách
- **Katalog stylů**
  - Pomáhá rozeznat společné vlastnosti stylů
  - Pomáhá kategorizovat do skupin se společnými vlastnostmi
    - Event systém je podtyp nezávislých komponent
    - Event systém má další dvě podkategorie
      - *Implicit invocation*
      - *Explicit invocation*
  - Říká za jakých okolností je vhodné styl použít

Architektonické styly - Katalog



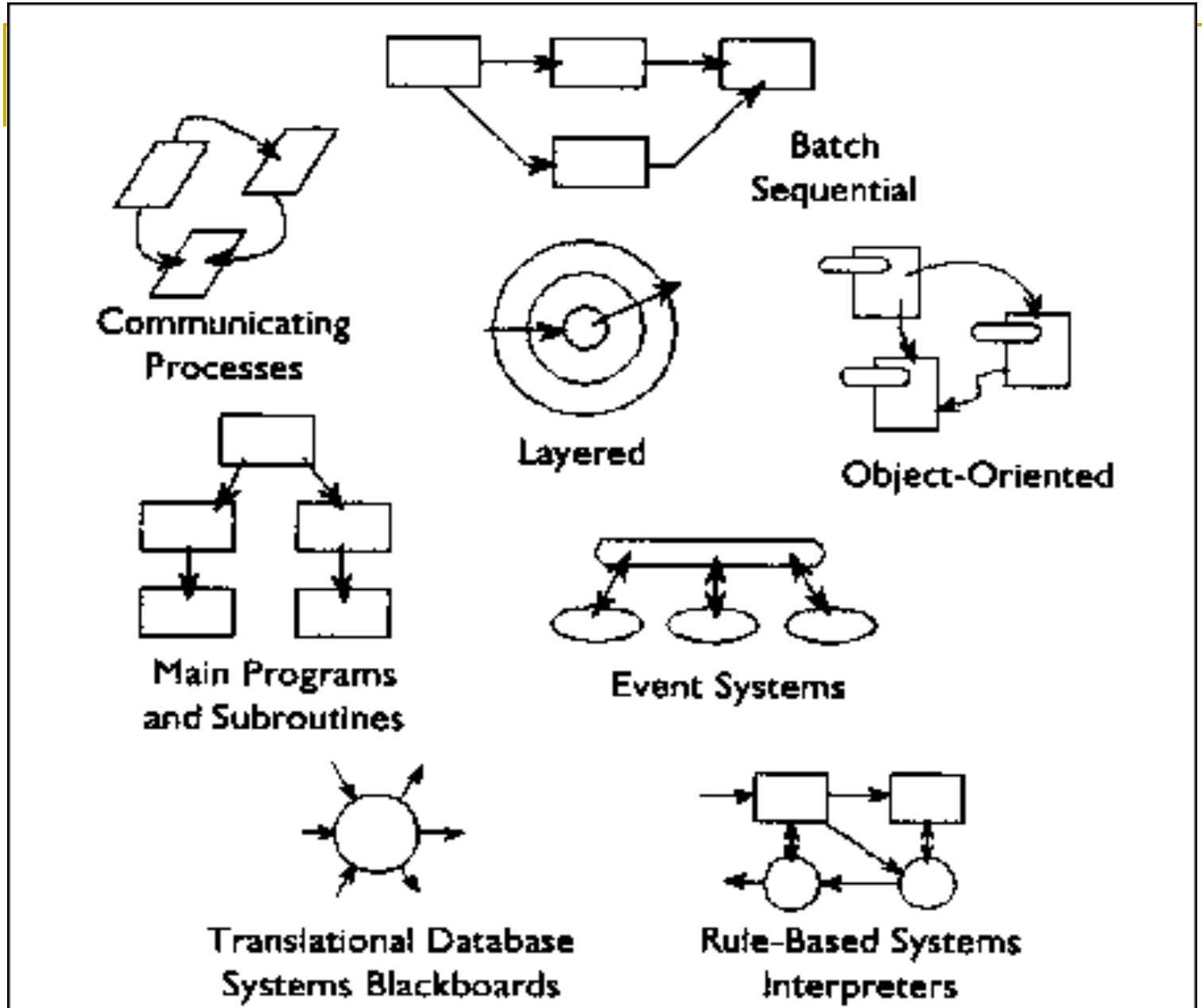


FIGURE 1: ARCHITECTURAL STYLES.

Styly

I. Datacentrické

II. Data-flow

III. Virtual Machine

IV. Call-n-return

V. Nezávislé komponenty

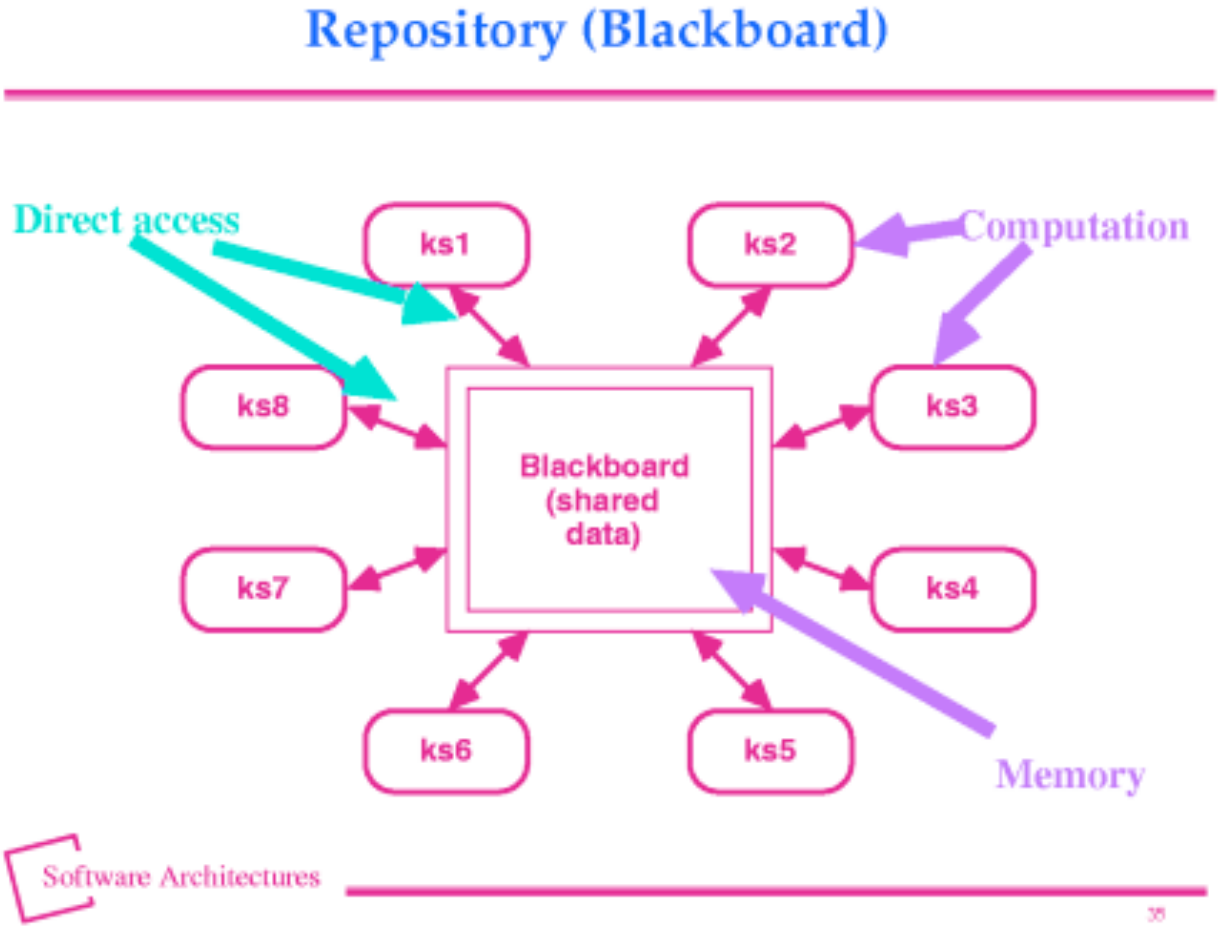
VI. Heterogenní

https://www.cs.cmu.edu/afs/cs/project/tinker-arch/www/html/Tutorial_Slides/Soft_Arch/base.021.html

I. Datacentrické architektury

- Cílem je dosažení kvality integrace dat
- Systémy kde je důležitý
  - přístup a změna k datovému skladu

I. Datacentrické architektury



I. Datacentrické architektury

- Klient je nezávislé vlákno s řízením
- Datová struktura je sdílena mnoha klienty
  - Ti přistupují k datům a případně je modifikují
  - **Globální** stav
- Rozdělme dále
  - **Pasivní** repozitář (soubor)
  - **Aktivní** repozitář (blackboard)

I. Datacentrické architektury

1. Pasivní repozitář

- Klient přistupuje k central. datové struktuře a mění ji.
- Centrální repozitář umožní přístup několika klientům
- Probe (sonda zda nastala změna)
    - Web aplikace
- Komunikace a koordinace udána klientem

2. Aktivní repozitář = **Blackboard**

- Blackboard zasílá notifikace pro subscribery
    - pokud se změní data o které má subscriber zájem
- Blackboard je aktivní
- Komunikace a koordinace obousměrně

I. Datacentrické architektury

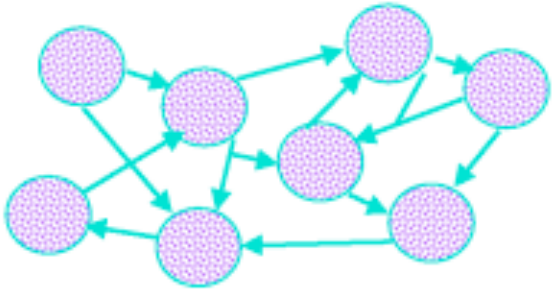
- Tento styl nabízí strukturní řešení pro **integraci**
- **Klienti** jsou **nezávislí** mezi sebou
- **Datastore** je **nezávislý** na klientech
- Styl má dobrou **škálovatelnost**
  - Snadno se přidá nový klient
- **Modifikace** klienta je snadná
  - Jeden klient nemá vazbu na ostatní
- **Coupling** klientů sníží modifikovatelnost, ač zvýší výkon
- Pokud je klient vlastní proces => styl *Klient/server*
    - Klient/server je sekce *: independent komponents*
    - Pozn. Styly nejsou přímo oddělené mezi sebou

II. Data-flow architektury

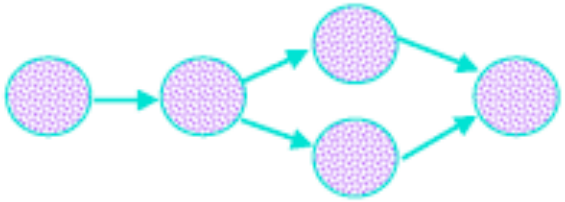
- Cílem je dosažení kvality
  - **znovupoužití** a **modifikovatelnosti**.
- Pohled na systém
  - Série transformací následných částí nad vstupními daty
  - Data vstupují do systému a tečou skrze komponenty, jedna podruhé tak, až se dostanou do cíle (output, data store)
- Styl má dva podtypy
  - **Batch sequential**
  - **Pipes & Filters**

II. Data-flow architektury typy

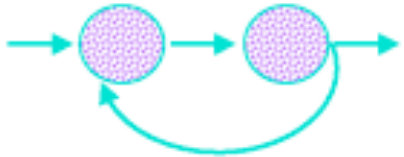
Kinds of Data Flow Systems



In general, data can flow in arbitrary patterns



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

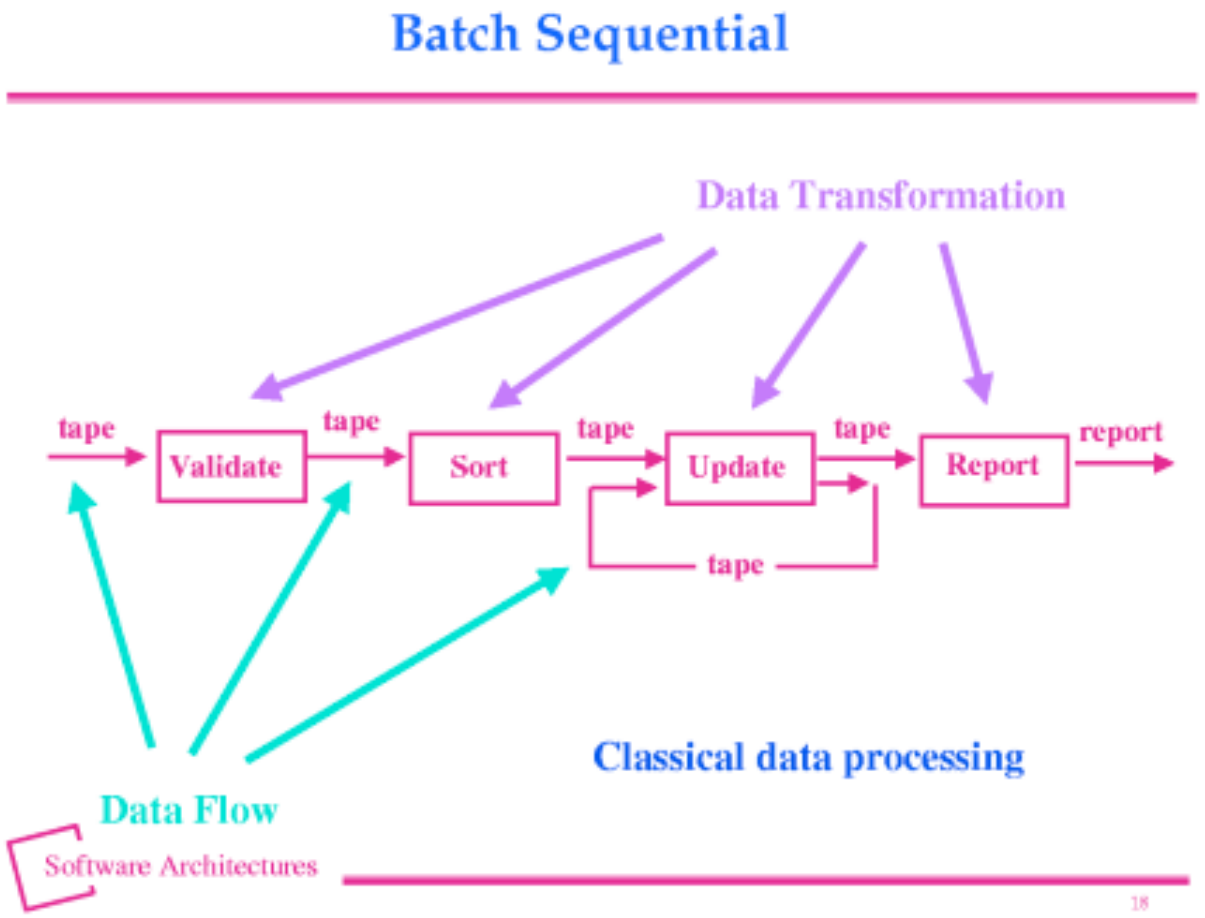


or in very simple, highly constrained cyclic structures

II. Data-flow architektury

- **Batch sequential**
  - Procesní kroky (**komponenty**)
    - nezávislé programy
  - Každý krok běží do konce předtím než poskytne data dál ke zpracování v dalším kroku
  - Každá batch data jsou odeslána jako celá část mezi kroky
  - Př.
    - Klasické zpracování dat

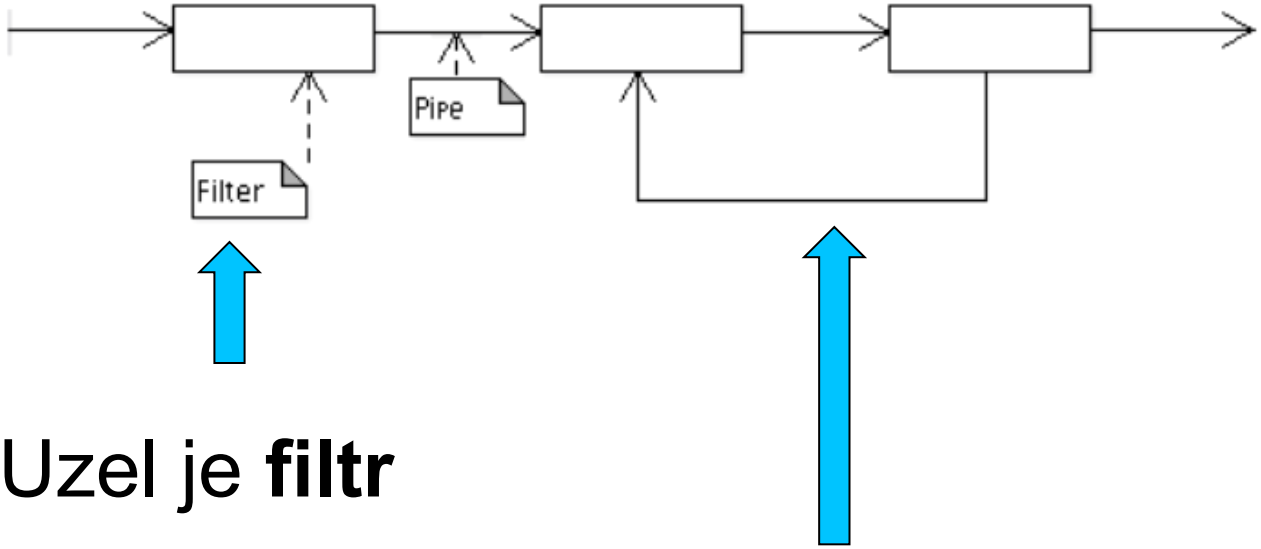
II. Data-flow architektury



II. Data-flow architektury

- **Pipe & filter**
  - Důraz na inkrementální transformace dat s následující komponentou
  - Typické pro UNIX programy
    - Inkremenálně transformují data
    - Používají kontext a nemají stav mezi instancemi
    - *ls -la* | *grep java*
  - **Pajpy** jsou bezestavové a existují aby přemístily data mezi **filtry**

II. Data-flow architektury



- Uzel je **filtr**
- Spojení mezi filtry je **roura**
  - Vede datový tok

II. Data-flow architektury

- Pajpy a filtry
  - Nedeterministické
  - Běží dokud jsou k dispozici další výpočet a přesuny
- Omezení indikuje spojení pro pajpy a filtry
  - Pajpa má zdrojový konec a výstupní konec
  - Zdrojový konec připojen k výstupnímu portu filtru
  - Výstupní konec připojen k vstupnímu portu filtru

II. Data-flow architektury - Pipe n Filters

- **Výhody** jsou především
  - Jednoduchost
    - Limitováno jak lze interagovat s prostředím
  - Snadná údržba a reuse
  - Filter = blackbox
  - Snadná hierarchie obou komponent
  - Spojení dvou filtrů tvoří navenek složitější filter a funkční blok
  - Zpracování dat je izolované
    - Snadná paralelizace a distribuce
    - Možnost zlepšení výkonu

II. Data-flow architektury - Pipe n Filters

- Mezi **nevýhody** patří
  - Vlastní struktura
    - znemožní aplikací na interaktivní aplikace
  - Seřazení filtrů je složité
  - Filtry nemohou kooperovat
  - Výkon je špatný
    - Izolace funkcionality
    - Nejnižší společný dělitel pro datovou reprezentaci
      - Ascii stream
    - ..

II. Data-flow architektury - Pipe n Filters

.. **nevýhody** ..

- Výkon je špatný
  - ..
  - Každý filtr musí mít parser
  - Pokud filtr nemůže produkovat výstup než načte vstup potom potřebuje veliký buffer
    - Sort
    - Omezený buffer = potenciální deadlock
  - Každý filtr je proces nebo procedure-call

III. Virtual machine architecture

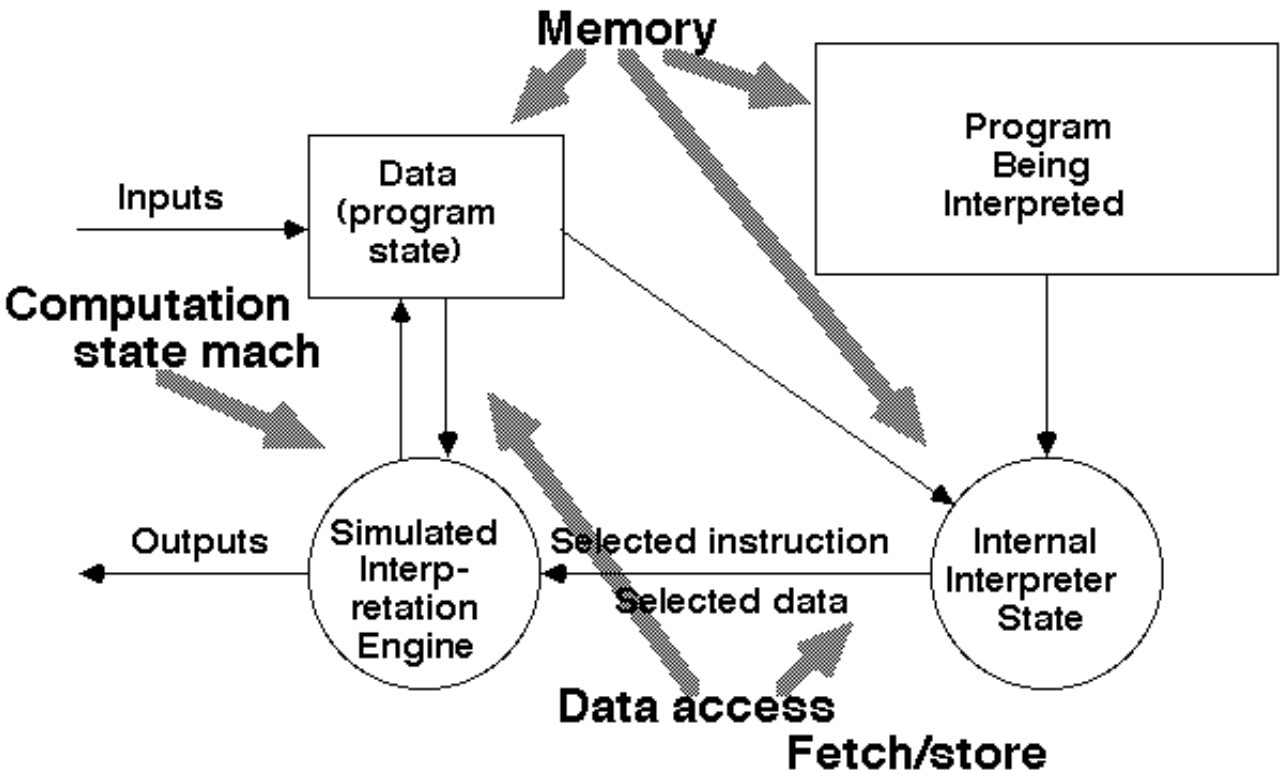
- Cílem je kvalita přenositelnosti.
- Simulují funkčnost která není nativní k HW nebo SW na kterých je implementována
- Vhodné pro
  - Simulace platformy, která není kompletní
  - Simulace „pohrom“
    - Bez simulace drahé, komplexní, nebezpečné

III. Virtual machine architecture

- Interpret
- Tabulkově řízený překladač
- Rule-based systém
- Syntaktický shell
- Command procesor
- Java napsána nad JVM
  - Platform independence

III. Virtual machine architecture

Interpreter



III. Virtual machine architecture

Pozn

- Interpretovaný program
- Programová data
- Vnitřní stav interpretru
- Překladové jádro
  - vybírá instrukce z interpretovaného programu
  - mění vnitřní stav
  - dle instrukce mění programová data

III. Virtual machine architecture

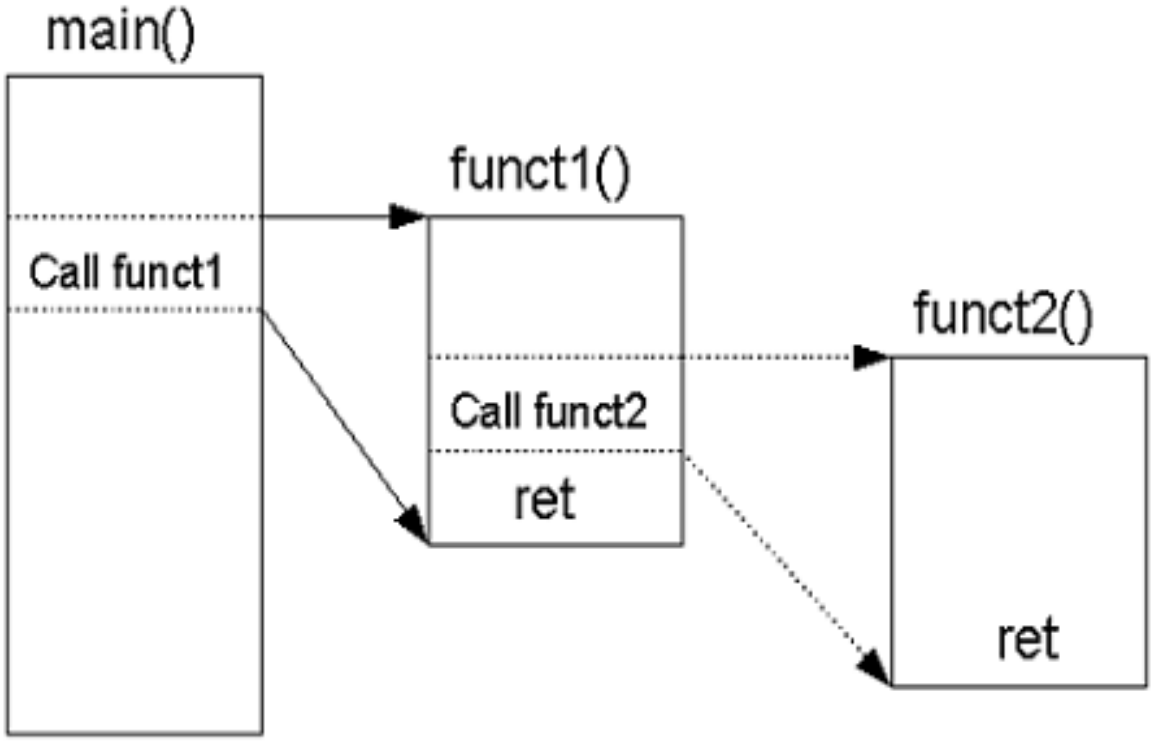
- Program běžící na interpretru přidává
  - Flexibilitu
  - Možnost přerušit
    - A dotazovat se na program
    - Měnit za běhu
  - Výkonnostní úbytek
    - Za další výpočty

IV. Call and Return Architektury volání s návratem

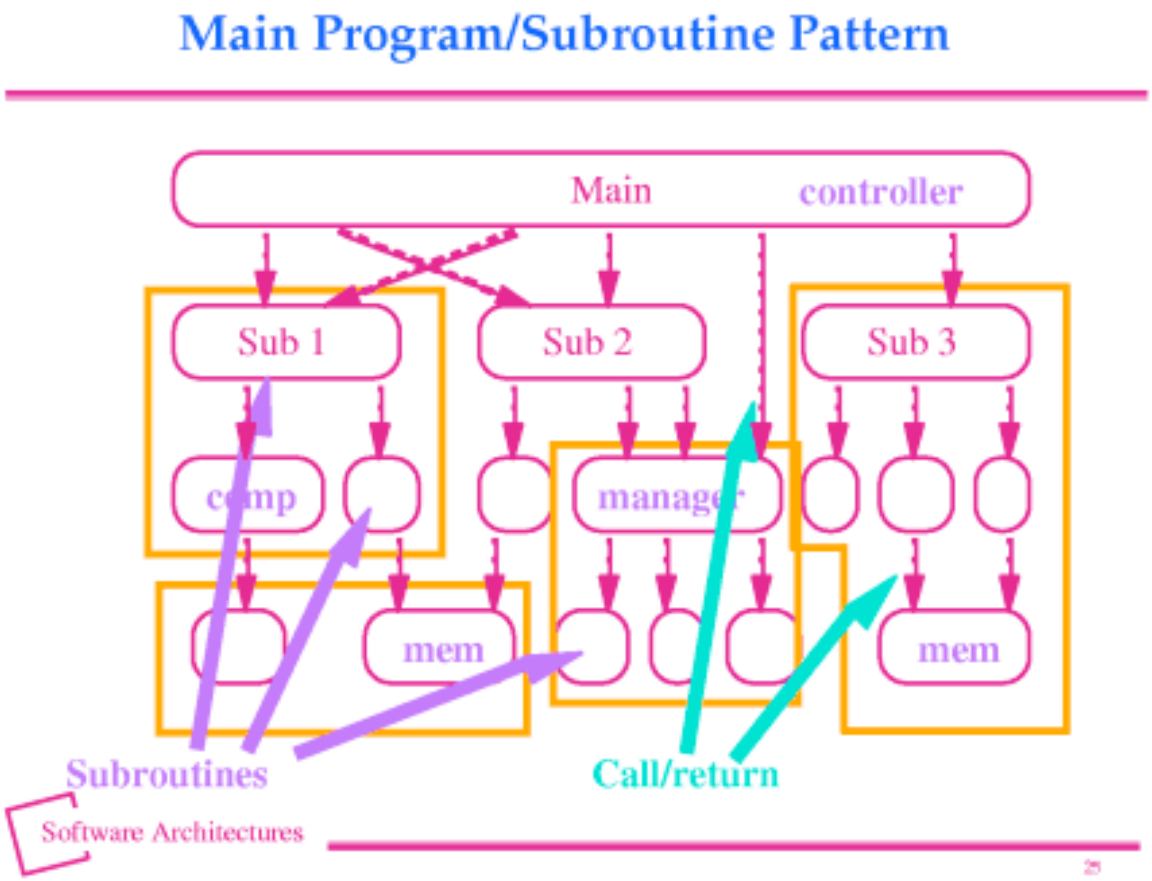
- Cílem je dosažení
  - **modifikovatelnosti** a **škálovatelnosti**.
- Dominantní architektury po 30let
- Široká škála pod stylů
  - Main program a subrutiny
  - Remote procedure call
  - Abstract data type
  - Object-oriented
  - Layered system

IV. Call and Return Architektury

- Main program a subrutiny



IV. Call and Return Architektury



IV. Call and Return Architektury

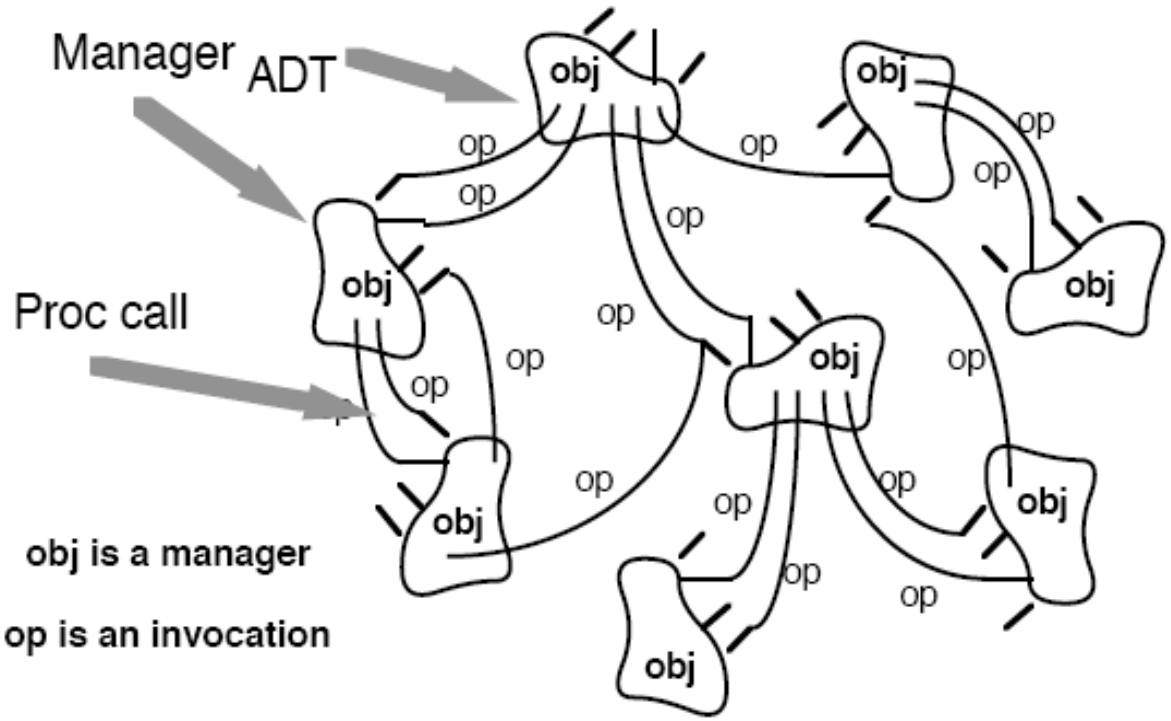
- Main program a subrutiny
  - Cílem je dekompozice programu na menší části se snazší modifikovatelností.
  - Program je dekomponován hierarchicky
  - Hlavní vlákno řízení a každá komponenta získává toto řízení od rodiče k potomku

IV. Call and Return Architektury

- Remote procedure call (RPC)
  - Jako main program a pod-rutina jsou dekomponovány na části, ale ty žijí na různých počítačích které jsou spojené po síti.
  - Cílem je zvýšení výkonu pomocí distribuce výpočtu mezi více procesorů
  - V RPC je přiřazení práce určeno v run-time
  - Mimo toho, že volání procedur může trvat déle jelikož voláme jiný počítač, není zde patrná jiná odlišnost od main program a pod rutina
    - Pozn. OOP a CORBA

IV. Call and Return Architektury

- **Object oriented** nebo **Abstract data type**



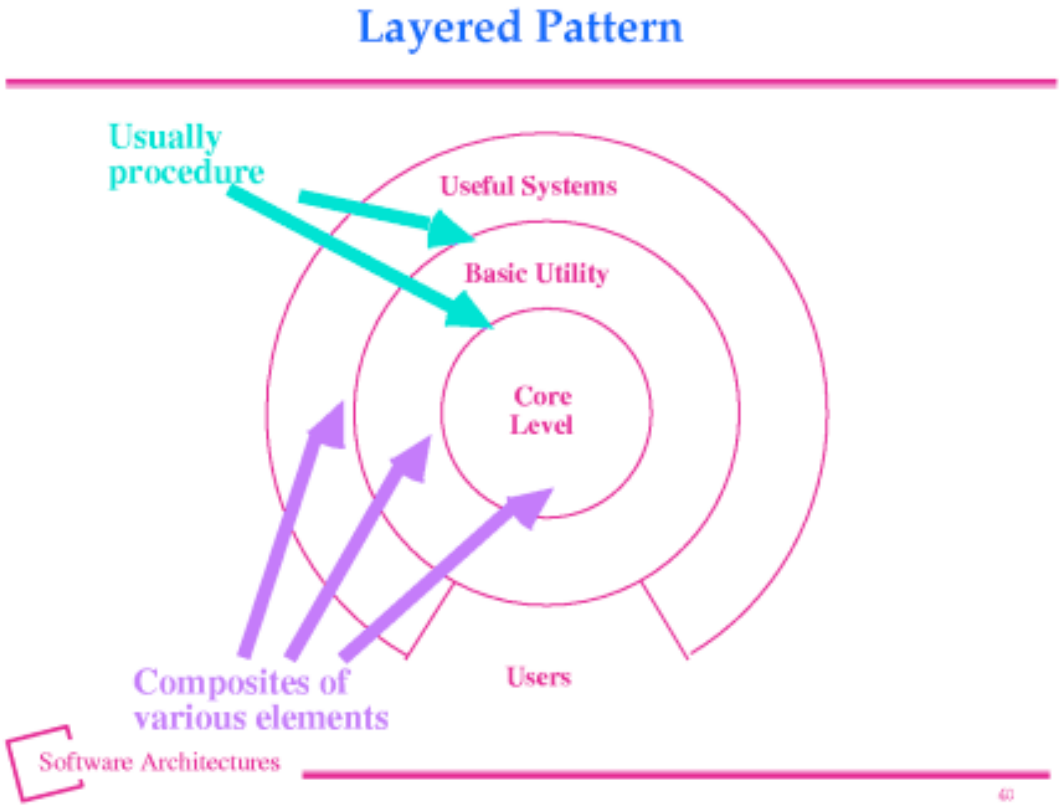
IV. Call and Return Architektury

- Object oriented / Abstract data type
  - Moderní verze call-and-return architektury
  - Záměrem je svázání dat a znalost manipulace a přístupu k datům
  - Cílem je dosažení
    - Modifikovatelnosti
    - Zapouzdření a schování interních detailů
  - Přístup k objektu přes operace, metody
  - Speciální omezeni na procedure call
  - Poskytuje reuse, modifikovatelnost a oddělení zájmů
  - Služby nevědí nic o vnitřní reprezentaci

IV. Call and Return Architektury

- Object oriented / Abstract data type
  - Hlavní charakteristiky **rozlišující OO a ADT**
    - object-oriented X abstract data type
      - **dědičnost a polymorfismus**
  - Pokud je abstrakce objektu black box služba a další komponenty požadující tyto služby pak hovoříme o **call-based client-server**

IV. Call and Return Architektury
Vrstevnatý systém



IV. Call and Return Architektury

- Vrstevnatý systém
  - Komponenty jsou rozmístěny ve vrstvách
    - Řídí mezi komponentovou interakci
  - Striktní verze:
    - Vrstva komunikuje jen s nejbližšími sousedy
  - Cílem je dosažení kvalitu modifikovatelnosti a portovatelnost
  - Nejnižší vrstva poskytuje základní funkčnost jako HW přístup a OS volání jádra
  - Následné vrstvy rozšiřují svého předchůdce, schovávají nižší vrstvy a poskytují služby vrstvě vyšší

IV. Call and Return Architektury
Vrstevnatý systém

- Vyšší vrstvy často poskytují virtuální stroj
  - Kompletní množina koherentní funkčnosti nad kterou může být napsána aplikace
- Běžné vrstevnaté systémy nejsou „čisté“
  - Funkčnost jedné vrstvy volá funkčnost vrstvy ne nejbližších vrstev
  - => layer bridging
- Layer bridging
  - Pro efektivitu
  - Zhoršuje model
  - Stále snadná portabilita ač horší než u „pure“

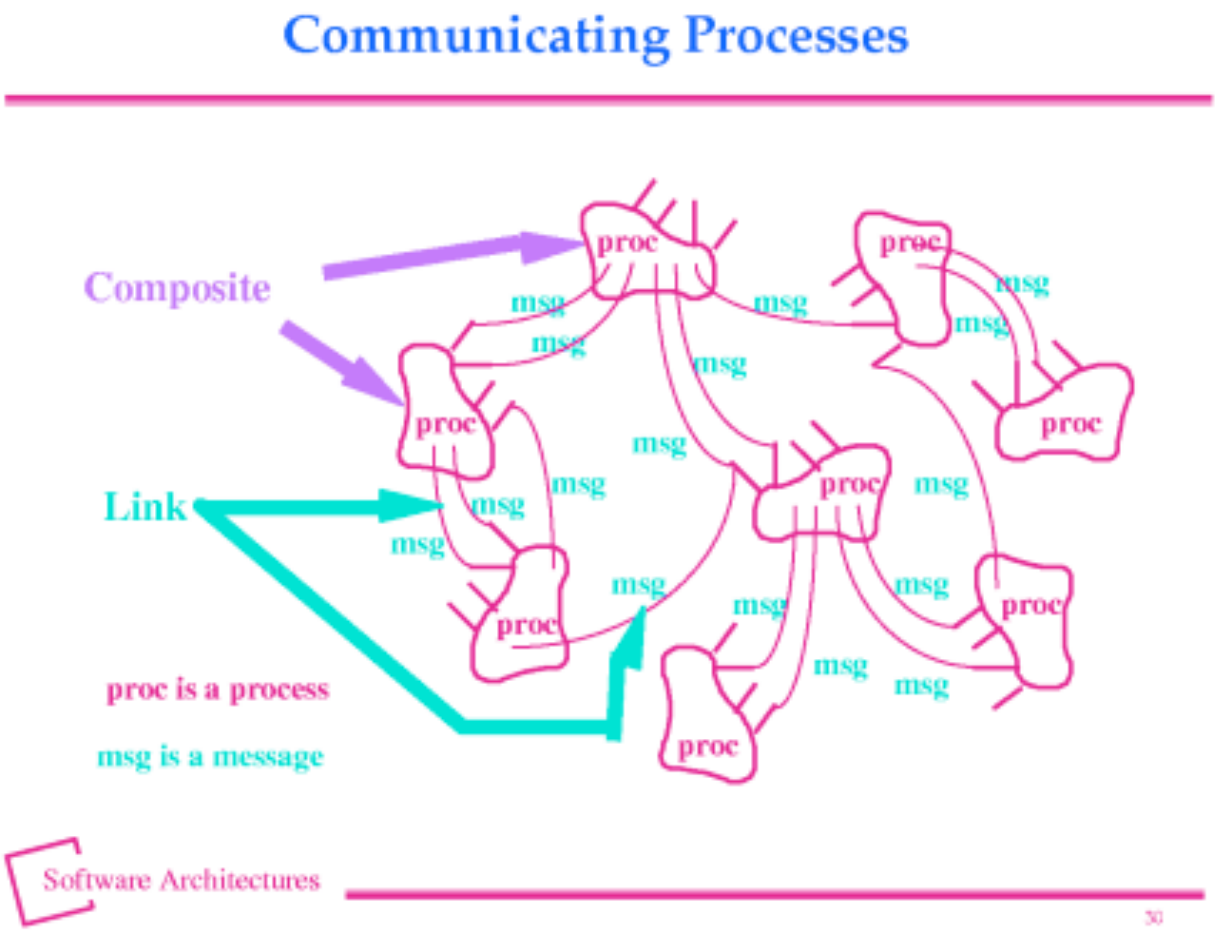
IV. Call and Return Architektury
Aspektově orientované pgm.

- Program rozdělen na běžnou a aspektovou část
- Samostatná údržba obou částí
- Propojení přes joint point
- Run-time vs statická kompilace
- Aspect weaver vplétá aspekty do běžné části
  - Pomoc při křížících se zájmech (výkon, bezpečnost)
  - Redukce kódu, maintenance, velikosti, zlepšení čitelnosti

V. Nezávislé komponenty

- Skládají se z
  - Několika nezávislých procesů
  - Nebo objektů které komunikují skrze zprávy
- Cílem je dosažení kvality **modifikovatelnosti**
  - Odtržení různých výpočtů
- Komponenty zasílají data mezi sebou, ale neřídí se navzájem
- Zpráva předána
  - Jmenným účastníkům
  - V případě event sytému pomocí subscribe/publish na nejmenné účastníky
- Implicitní x Explicitní invokace

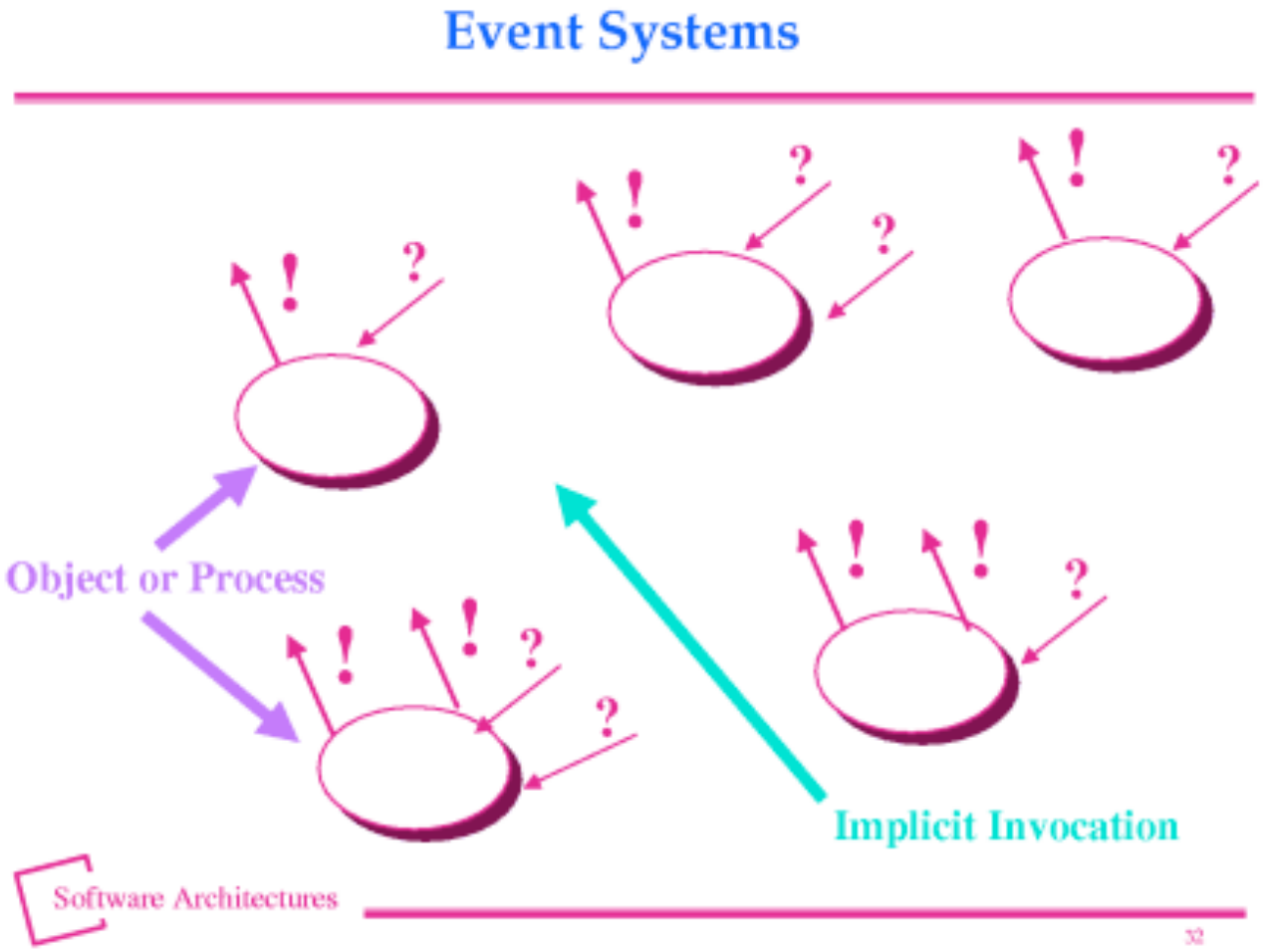
V. Nezávislé komponenty



V. Nezávislé komponenty

- **Událostní systém**
  - Podstyl kde je řízení částí modelu
  - Komponenta ohlásí data k publikaci
  - Ostatní komponenty registrují svůj zájem v daná data - *subscribe*
  - Pokud je poslouchající komponenta zaregistrována pak jsou data předána
  - Typicky existuje **message manager**, který řídí komunikaci mezi komponentami
    - Vyvolá komponentu když dorazí data
    - Př. **Observer** a **Mediátor**

V. Nezávislé komponenty



V. Nezávislé komponenty

- Událostní systém
  - Message manager
    - může ale nemusí řídit komponenty, kterým předává zprávy
  - Komponenty registrují
    - informace které poskytují
    - a které chtějí získávat
  - Komponenty publikují informace
    - zasláním zprav message manageru
    - ten přepošle zprávy dále

V. Nezávislé komponenty

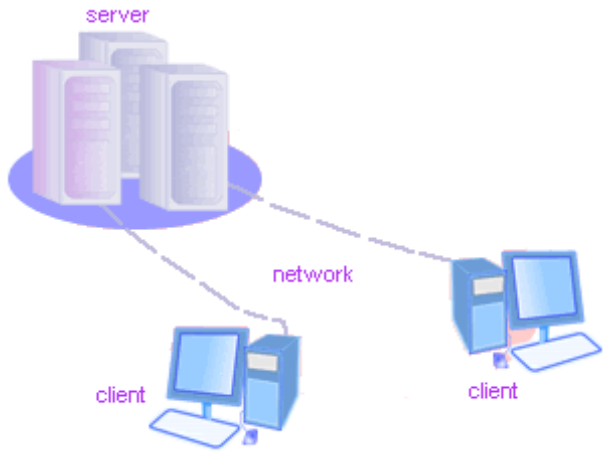
- Událostní systém
  - Odpáruje implementaci komponent od znalosti dalších komponent (jmen, umístění)
  - Může zahrnovat odpárování kontroly
    - Paralelní výpočet
    - Interakce skrze výměnu dat
  - Usnadní integraci komponent

V. Nezávislé komponenty

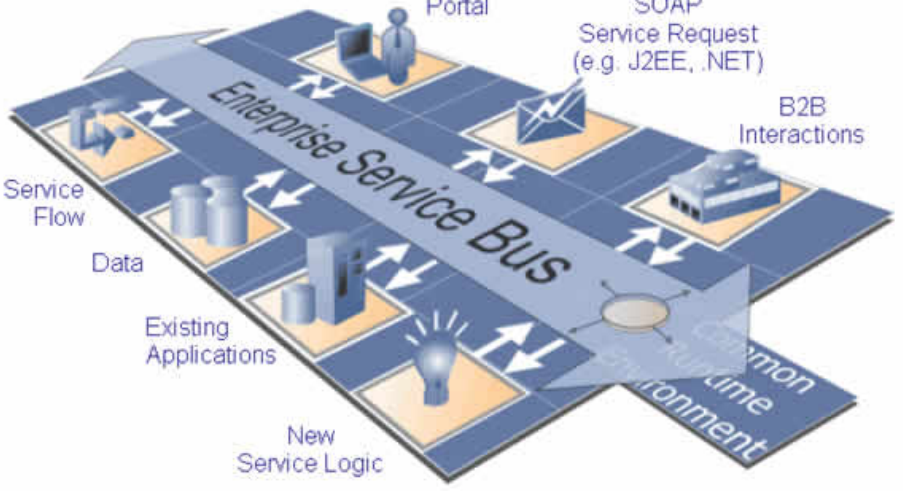
- **Komunikující procesy**
  - Pod-styl
  - Multi-procesorové systémy
    - Klient-server
  - Cílem je dosažení kvality škálovatelnosti
  - Server existuje aby dodával data na jednoho či více klientů
    - Ti na různých místech

V. Nezávislé komponenty

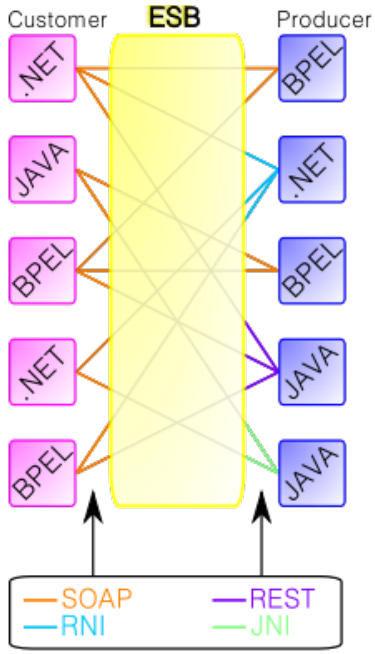
- **Komunikující procesy**
  - Klient začíná zasláním požadavku na server
    - Ten pak synchronně či asynchronně odpoví
    - Synchronně
      - Vrací data a řízení ve stejný čas
    - Asynchronně
      - Vrací jen data
      - Vlastní vlákna



V. Nezávislé komponenty



- Enterprise Service Bus
  - Návrh interakce a komunikace mezi SW aplikacemi v Service Oriented Architecture
  - Značně rozšířená varianta Client-server
  - Asynchroní zprávy
  - Cíl: Integrace heterogeních systémů, malý coupling
  - Základní prvek je zběrnice (Bus)
    - Monitoruje a řídí zprávy
    - Řídí nasazení a verze
    - Umožní použití záložních služeb
  - Byznys procesy a služby
    - posány přes **Business Process Execution Language** (BPEL)



VI. Heterogenní styly

3 typy:

- Lokálně heterogenní
  - Run-time struktura obsahuje vzory z různých stylů v různých částech
    - Main-program-sub-routine má datový repozitář
- Hierarchicky heterogenní
  - Koncept komponenty jednoho stylu je po dekompozici strukturován v jiném stylu
- Současně heterogenní
  - Více stylů současně popisuje systém

Comparison of System Patterns

System Model	Components	Connections	Control Struct
Pipeline			
stream -> stream	filters (local processing)	data flow ASCII streams	data flow
Data abstraction (object-oriented)			
localized state maint	servers (ADTs, objs)	procedure call	decentralized, single thread
Events			
implicit invocation	independent components	blind announce	loose coupling
Interpreter			
virtual machine	state mach, two memories	fetch, store	input-driven
Repository			
central database	1 memory N processes	direct access or proc call	internal or external

Common Architectural Idioms

1. Data flow systems
   - Batch sequential
   - Pipes and filters
   - Control loops
2. Call-and-return systems
   - Main program & subroutines
   - Object-oriented systems
3. Independent components
   - Communicating processes
   - Event systems
4. Data-centered systems (repositories)
   - Databases
   - Blackboards
5. Virtual machines
   - Interpreters
   - Rule-based systems
   - Hierarchical layers

... and more ...

Organizace stylů

- Vztahy mezi styly
  - Hlavní rozdíly pro konkrétní úkoly
  - Styl jako varianta jiného stylu
    - Kombinace
  - Návrhové problémy spjaté s klasifikací vzorů

Organizace stylů - kategorie

- Charakter **interakce** komponent
  - **Kontrolní řízení**
  - **Datová volání**
- Přesněji
  - Jak je kontrola
    - sdílena, alokována, přesunuta mezi komponenty
  - Jak data komunikují v systému
  - Jak data a kontrola spolupracují
  - Co styl dovolí
  - Typy komponent a konektorů ve stylu

Organizace stylů - Komponenty a konektory

- **Komponenty a konektory**
  - Primární stavební blok
  - Komponenta je jednotka SW
    - Vykonává funkce v runtime
    - Programy, objekty, procesy a filtry
  - Konektor
    - Mechanismus k řízení komunikace, koordinace, kooperace.
    - Sdílená reprezentace, RPC, message-passing protokol, data stream
  - Selekce částí neidentifikuje styl

Organizace stylů – Kontrola/řízení

- **Kontrola / Řízení**
  - Kontrola mezi komponentami
  - Spolupráce komponent
  - Obsahuje
    - **Topologii**
    - **Synchronicitu**
    - **Binding time**

Organizace stylů – Kontrola/řízení

- **Topologie**
  - Jakou geometrickou podobu má kontrolní tok
    - Pipeline je **lineární** či **acyklická**
    - Main-program a sub-routine je hierarchická **stromová**
    - Server má **hvězdici**
    - Sekvenční procesy mají **libovolnou** topologii
  - Zajímá nás také **směr** toku
  - Topologie je **statická** nebo **dynamická**

Organizace stylů – Kontrola/řízení

- **Synchronicita**
  - Jak jsou závislé **akce** na kontrolních **stavech**.
    - Batch sequential nezačne dokud nezačal předchůdce
    - SIMD a MIMD pro paralelní výpočty
  - Synchronní systém
    - Komponenty se synchronizují často
    - Stavy nepredikovatelné
  - Asynchronní
    - Komponenty jsou nepredikovatelné v interakci
    - Synchronizace jednou za čas
      - **Bariera**, **paralelní redukce**
    - Autonomní systémy pracují nezávisle a paralelně

Organizace stylů – Kontrola/řízení

- **Binding time**
  - Kdy je zahájen přesun řízení
  - Někdy určeno programem
    - Write time
    - Compile time
    - Invocation time
    - Nebo Dynamicky v runtime

Organizace stylů - Data

- **Topologie**
  - Geometrický tvar systému z hlediska datových toků.
  - Podobně jako topologie kontroly
- **Kontinuita**
  - Jak spojitě jdou data skrze systém
  - Spojitá
  - Sporadická – diskrétní
  - High volume
  - Low volume

Organizace stylů - Data

- **Mode**
  - Dostupnost dat v systému
    - OO předává z komponenty na komponentu
    - Shared data
    - Modifikace a znovu vložení
      - Copy out copy in
    - Broadcast
    - Multicast
    - Unicast
- **Bind time**
  - Kdy je zahájen přenos

Organizace stylů – Kontrolní a Datová spolupráce

- **Tvar**
  - Jsou topologie kontroly a dat částečně izomorfní?
- **Směr**
  - Má kontrolní tok stejný směr jako to datový či opačný?
    - Pipe n Filter
      - Stejný
    - Client/server
      - Opačný
        - » Kontrola na server z klienta
        - » Data ze server na klienta

Organizace stylů – Typové úvahy

- Různé třídy architektur mají různou analýzu
  - Asynchronní paralelní komponenty
    - Nondeterministic state machine
  - Fixní sekvence
    - Dělení funkcí
- Analýza podstruktur
  - Kombinace

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	Sporadic	Yes
Communicating Processes	Processes	Message Protocols	Arbitrary	Any but Sequential	Arbitrary	Sporadic	Possibly
Data Centered							
Repository	Memory, Computations	Queries	Star	Asynchronous	Star	Sporadic	Possibly
Blackboard	Memory, Components	Direct Access	Star	Asynchronous	Star	Sporadic	No

Table 1: Specializations of the dataflow network 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
Data flow styles: Styles dominated by motion of data through the system, with no "upstream" content control by recipient											
Dataflow network [B+88]	transducers	data stream	arbitrary	asynch	i, r	arbitrary	cont lvol or hvol	passed	i, r	yes	same
• Acyclic [A+95]			acyclic			acyclic					
• Fanout [A+95]			hierarchy			hierarchy					
• Pipeline [DG90, Se88, A+95]			linear			linear					
-Unix pipes and filters [Ba86a]		ascii stream			i				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)										

Table 2: Specializations of the interacting processes style

Style	Constituent parts		Control issues			Data issues				Ctrl/data interaction	
	Comp-onents	Connec-tors	Topo-logy	Synch-ronicity	Bind-ing time	Topo-logy	Contin-uity	Mode	Bind-ing time	Isomor-phic shapes	Flow dir-ections
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 iso-morphic 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			incom-plete graph	asynch		incom-plete graph		passed		yes	same
Broadcast			arb	asynch		star		bdcast		no	same
Token passing			arb	asynch		arb.		passed		yes	same
Decentralized servers											
Replicated workers			hier	synch		hier		passed shared		yes	yes
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), asynch (asynchro-nous), opp (opportunistic)										
Binding time	w (write-time--that is, in source code), c (compile-time), i (invocation-time), r (run-time)										
Continuity	spor (sporadic), lvol (low-volume)										
Mode	shared, passed, bdcast (broadcast), mcast (multicast), ci/co (copy-in/copy-out)										

Key Word in Context

Vyhledávací systém

Vstup:

Seřazená množina *řádků*
Řádek je seřazená množina *slov*
Slovo je seřazená množina *znaků*

Výstup:

seznam všech cyklicky posunutých řádků v abecedním pořadí

Key Word in Context

Úloha je učební nástroj

Umožňuje poukázat na efekty různých arch. stylů

Key Word in Context

Vstup:

FEL Architektury Soft. Systémů
Key Word in Context

Výstup:

Architektury Soft. Systémů FEL
Context Key Word in
FEL Architektury Soft. Systémů
in Context Key Word
Key Word in Context
Soft. Systémů FEL Architektury
Systémů FEL Architektury Soft.
Word in Context Key

Key Word in Context

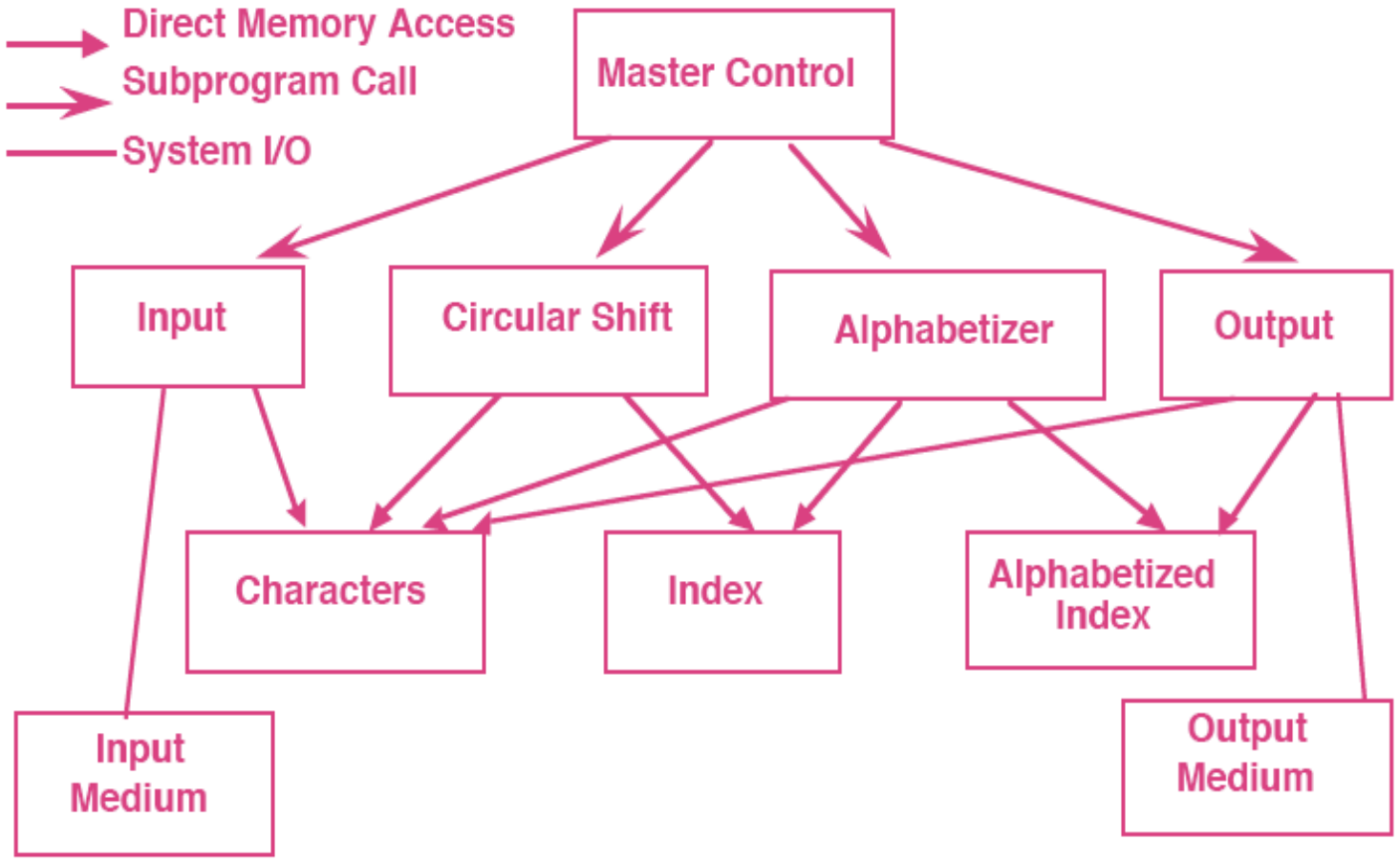
Kritéria:

- Změny v algoritmu – jak posunujeme
- Změny v datové reprezentaci – jak uloženy řádky
- Vylepšení funkcí – interaktivní systém
- Výkonnost – časová a paměťová
- Opětovné použití v jiných systémech

Problém lze řešit například pomocí:

- Hlavního programu a podprogramů se sdílenou pamětí.
- Abstraktním datovým typem – ADT
- Událostmi řízený systém
- Roury a filtry

Hlavního programu a podprogramy sdílená paměť



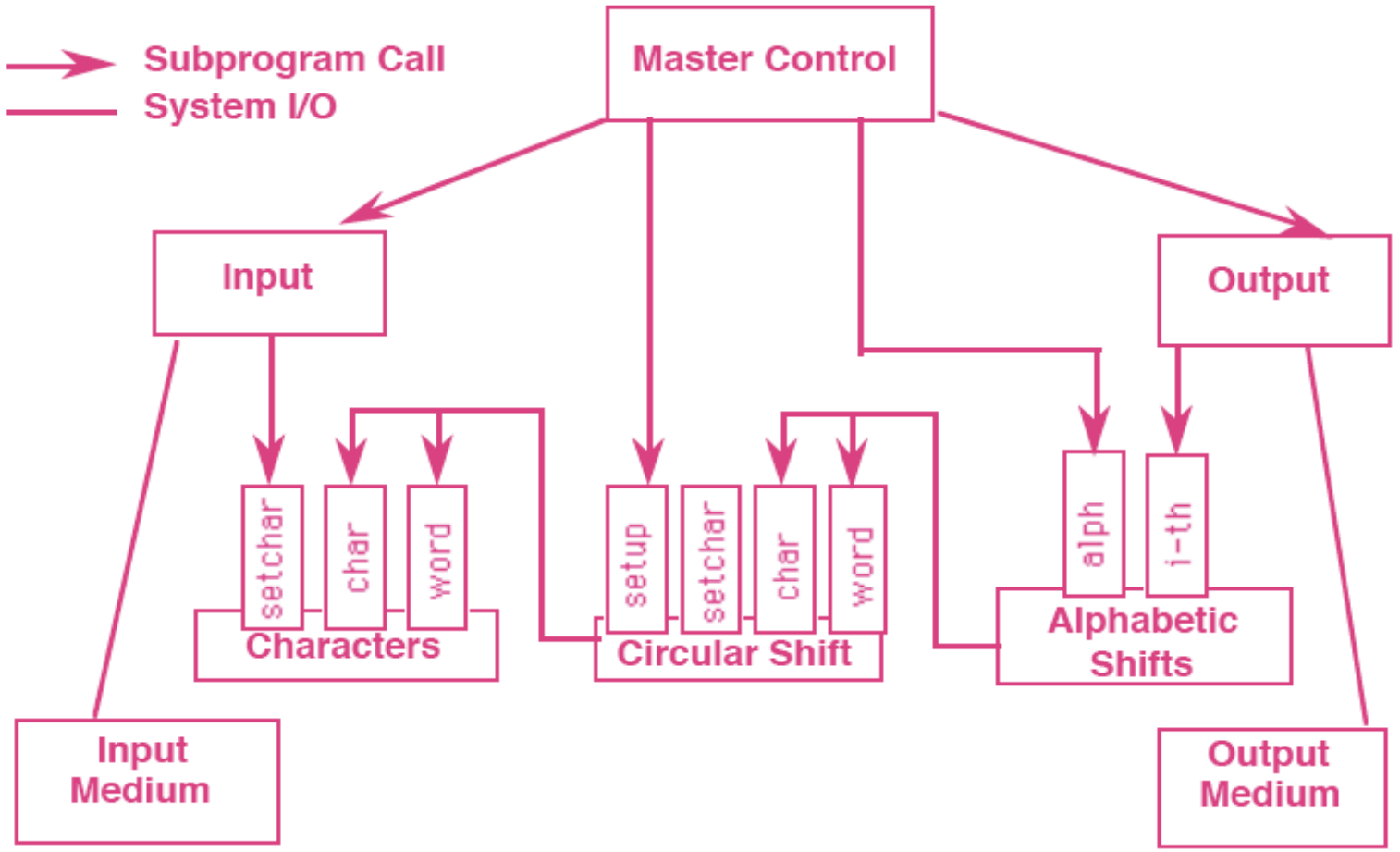
Key Word in Context

	Sdílená paměť	ADT	Události	Roury a filtry
Změny v algoritmu	?			
Změny v datové části	?			
Vylepšení funkcí	?			
Výkonnost	?			
Re-use	?			

Key Word in Context

	Sdílená paměť	ADT	Události	Roury a filtry
Změny v algoritmu	-			
Změny v datové části	-			
Vylepšení funkcí	-(+)			
Výkonnost	+			
Re-use	-			

Abstraktní datový typ



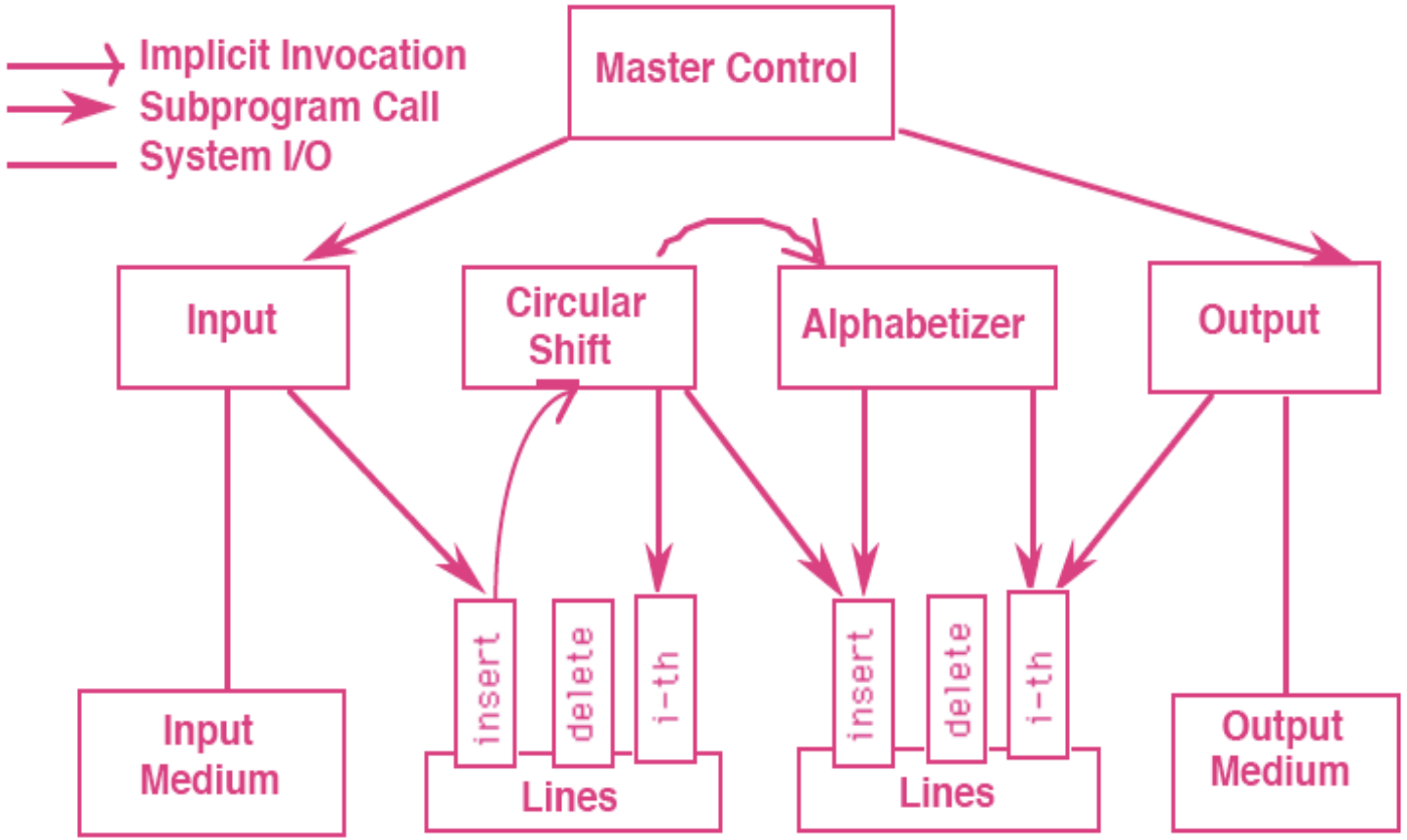
Key Word in Context

	Sdílená paměť	ADT	Události	Roury a filtry
Změny v algoritmu	-	?		
Změny v datové části	-	?		
Vylepšení funkcí	-(+)	?		
Výkonnost	+	?		
Re-use	-	?		

Key Word in Context

	Sdílená paměť	ADT	Události	Roury a filtry
Změny v algoritmu	-	-		
Změny v datové části	-	+		
Vylepšení funkcí	-(+)	-		
Výkonnost	+	+		
Re-use	-	+		

Událostmi řízený systém



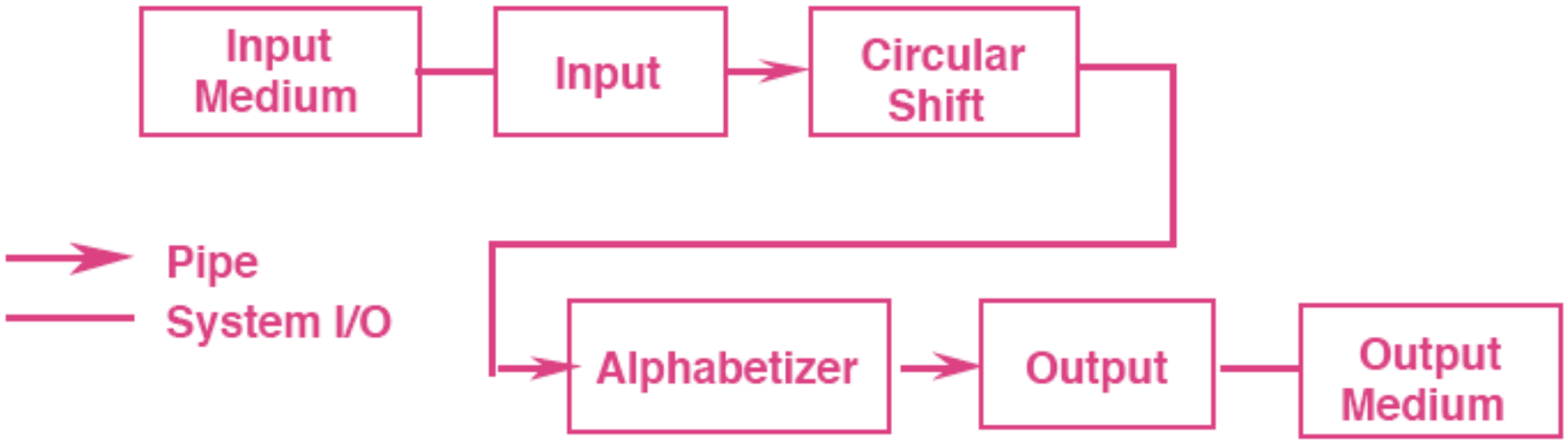
Key Word in Context

	Sdílená paměť	ADT	Události	Roury a filtry
Změny v algoritmu	-	-	?	
Změny v datové části	-	+	?	
Vylepšení funkcí	-(+)	-	?	
Výkonnost	+	+	?	
Re-use	-	+	?	

Key Word in Context

	Sdílená paměť	ADT	Události	Roury a filtry
Změny v algoritmu	-	-	+	
Změny v datové části	-	+	-	
Vylepšení funkcí	-(+)	-	+	
Výkonnost	+	+	-	
Re-use	-	+	-(+)	

Roury a Filtry



Key Word in Context

	Sdílená paměť	ADT	Události	Roury a filtry
Změny v algoritmu	-	-	+	?
Změny v datové části	-	+	-	?
Vylepšení funkcí	-(+)	-	+	?
Výkonnost	+	+	-	?
Re-use	-	+	-(+)	?

Key Word in Context

	Sdílená paměť	ADT	Události	Roury a filtry
Změny v algoritmu	-	-	+	+
Změny v datové části	-	+	-	-
Vylepšení funkcí	-(+)	-	+	+
Výkonnost	+	+	-	-
Re-use	-	+	-(+)	+