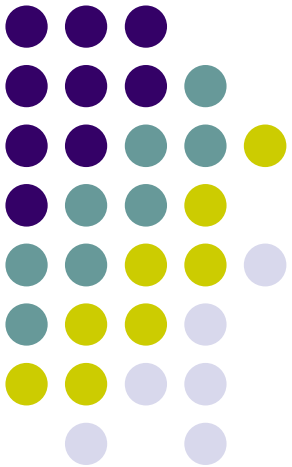
Padrões Arquiteturais

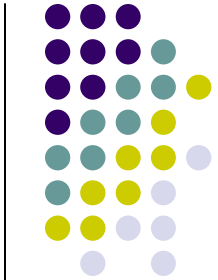
Pattern-Oriented Software Architecture (POSA)

Prof. Alberto Costa Neto

DComp/UFS

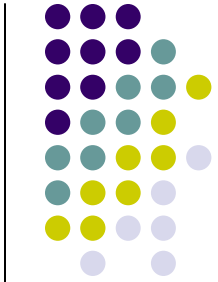


Padrões Arquiteturais

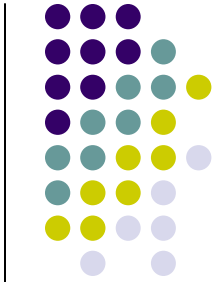


- Expressam esquemas fundamentais de organização estrutural do software.
- Provêem um conjunto de subsistemas predefinidos, especificam suas responsabilidades e relatam regras e guias para organizar as relações entre eles
- Padrões de alto nível (top level)
- Ajudam a alcançar propriedades globais, como:
  - Adaptabilidade da GUI
  - Portabilidade
  - ...

Categorização dos padrões arquiteturais (POSA)

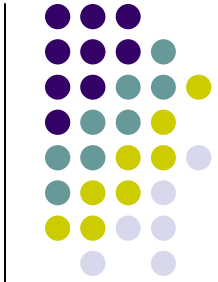


- Estruturação (From Mud to Structure).
  - Camadas, Pipes & Filtros e Quadro Negro.
- Sistemas Distribuídos
  - Broker
  - Referencia Microkernel e Pipes & Filtros
- Sistemas Interativos
  - Model-View-Controller e Presentation-Abstraction-Control
- Sistemas Adaptativos
  - Reflection e Microkernel

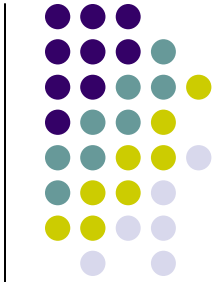


- Sistemas são estruturados usando vários padrões arquiteturais
  - Requerimentos diversos capturados por arquiteturas diferentes
  - Exemplo: distribuição e flexibilidade da GUI
- No entanto, um padrão particular ou uma combinação de vários não é a arquitetura de software completa
  - É um framework estrutural que deve ser refinado e especificado com mais detalhe:
    - Integração da funcionalidade com o framework
    - Detalhe de componentes e relacionamentos (talvez, usando padrões)

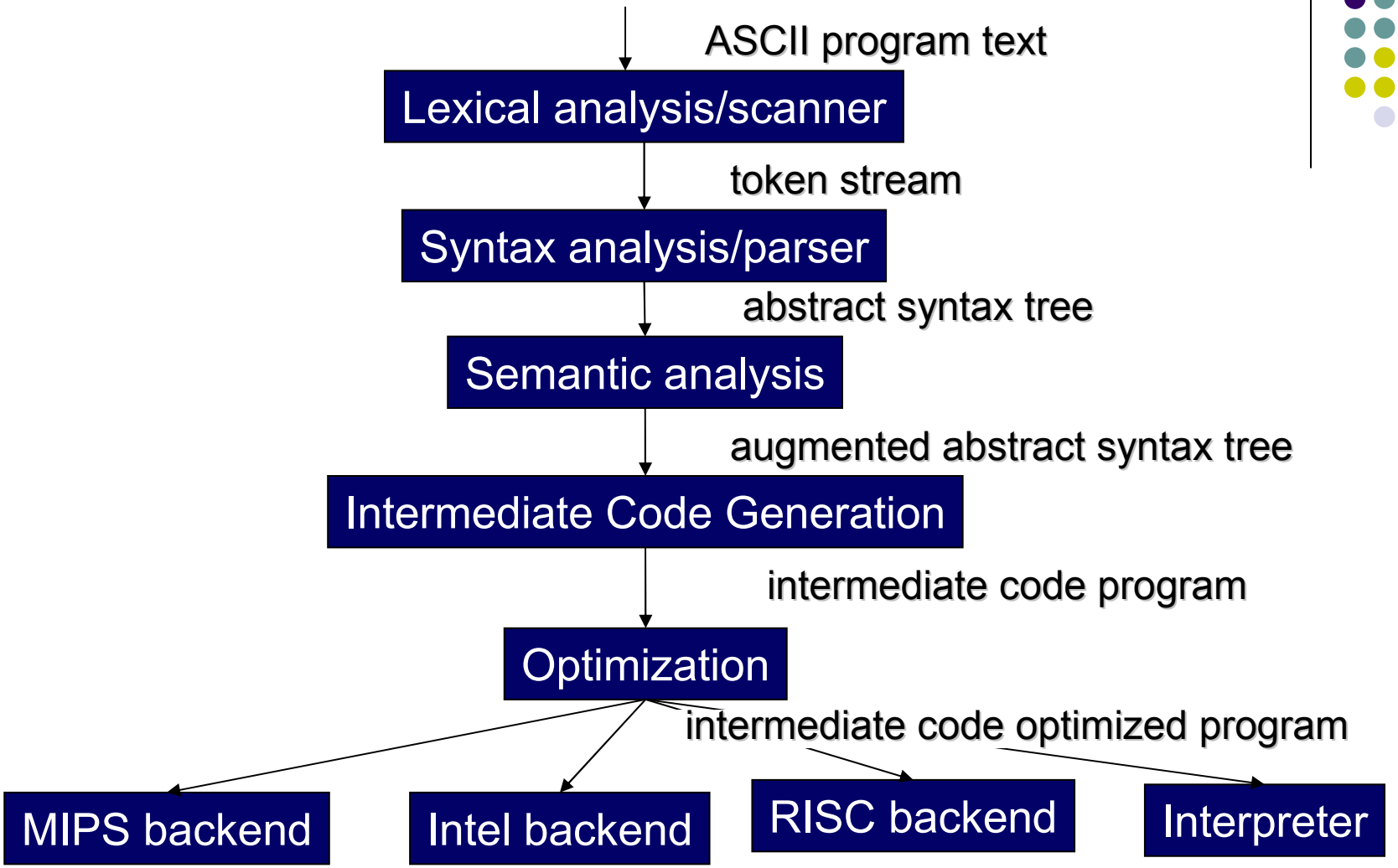
Exemplo de padrão Arquitetural Pipes e Filtros (POSA)

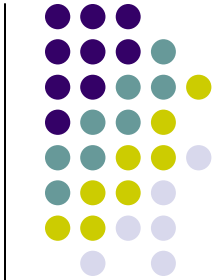


- Provê uma estrutura do sistema que processa um fluxo de dados.
- Cada passo de processamento é encapsulado num componente filtro.
- Dados são passados através de pipes entre filtros adjacentes.
- A recombinação de filtros permite construir famílias de sistemas relacionados.

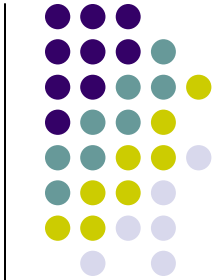


- Exemplo
  - O compilador de uma linguagem de programação
  - Perseguimos portabilidade de plataforma de hardware
    - Usamos uma linguagem intermediária

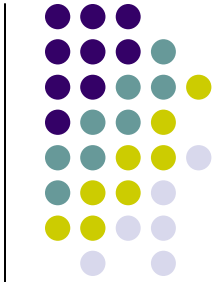

ASCII program text
Lexical analysis/scanner
token stream
Syntax analysis/parser
abstract syntax tree
Semantic analysis
augmented abstract syntax tree
Intermediate Code Generation
intermediate code program
Optimization
intermediate code optimized program
MIPS backend
Intel backend
RISC backend
Interpreter



- Contexto: Processamento de fluxo de dados (data stream)
- Problema: construir um sistema que processa ou transforma um fluxo de dados. Não pode ser feito num único componente.
  - Vários desenvolvedores
  - A tarefa é decomposta naturalmente em várias etapas de processamento
  - Existe probabilidade de mudanças (sistema flexível)



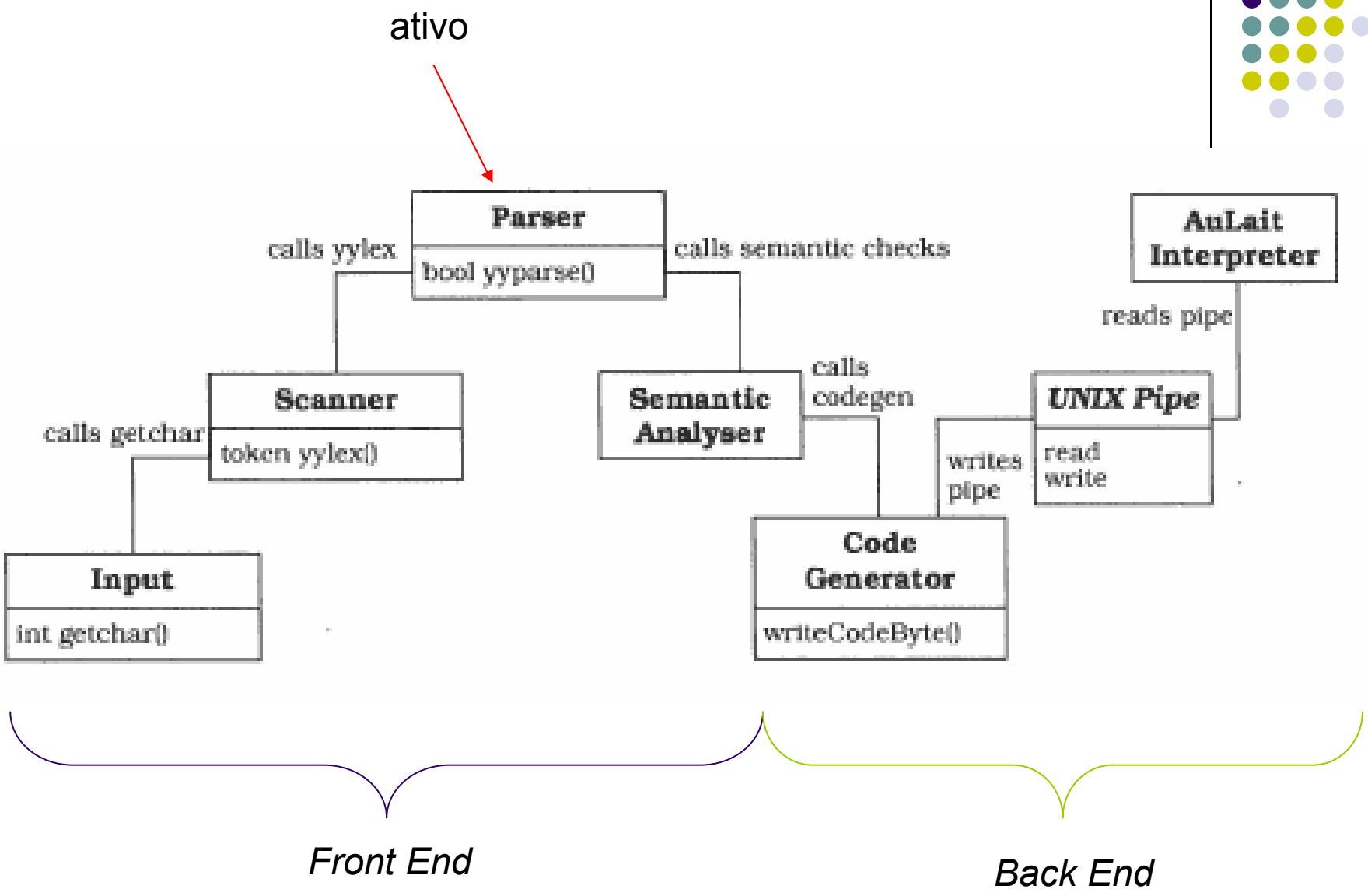
- As seguintes forças devem ser consideradas:
  - Melhoras futuras devem ser possíveis trocando um componente ou combinando vários, ainda por usuários
  - Componentes pequenos são mais adequados para reuso e outros contextos
  - Diferentes fontes de dados
  - Apresentação de resultados de diversas maneiras
  - Armazenamento explícito de resultados intermediários
  - Multi-processamento



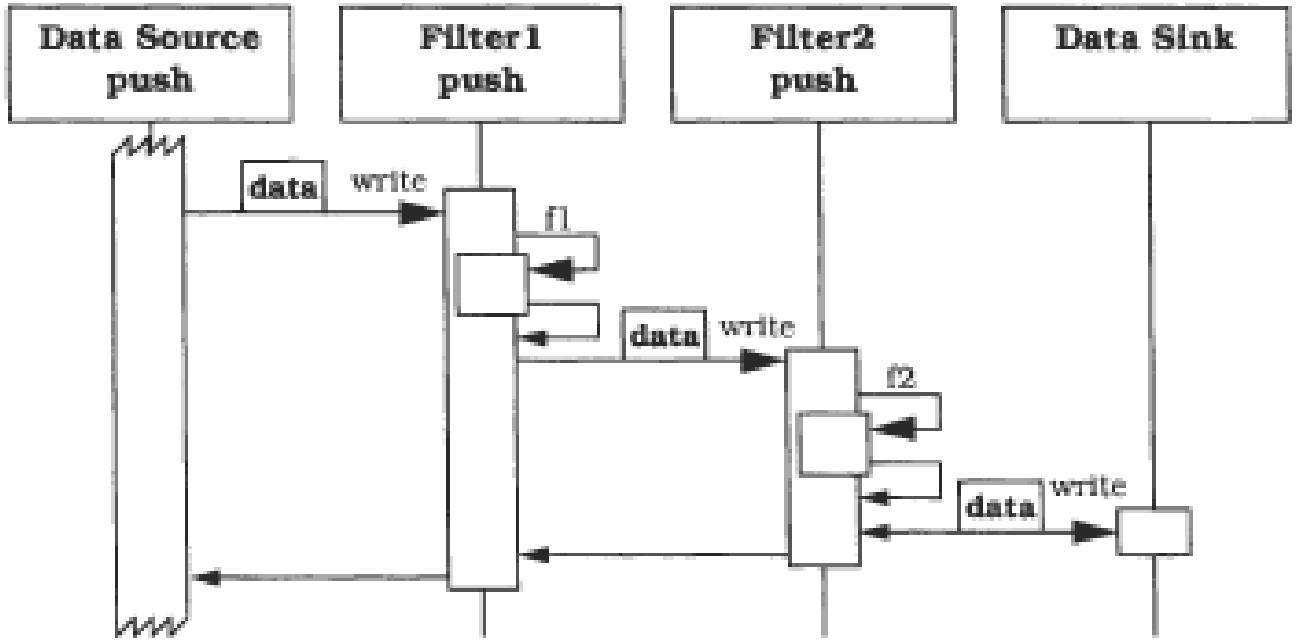
- Solução:
  - Dividir a tarefa do sistemas em várias etapas de processamento seqüencial (Filtros)
  - Etapas conectadas (por pipes) pelo fluxo de dados
  - Filtros produzem e consumem dados de maneira incremental, habilitando a exploração de paralelismo real
  - A seqüência de filtros combinados com pipes é chamada pipeline de processamento

- Estrutura
  - Filtros são as unidades de processamento do pipeline
  - Filtro enriquece, refina ou transforma a entrada de dados
  - Atividades são disparadas por:
    - O filtro subseqüente puxa dados
    - O filtro anterior põe dados

    } Filtros passivos
    - O filtro é um loop **ativo**, pondo e puxando dados
  - Pipes são conexões entre filtros
    - Pipes entre filtros ativos são buffers LIFO
    - Chamada a função ou procedimento, se a atividade é controlada por um filtro adjacente

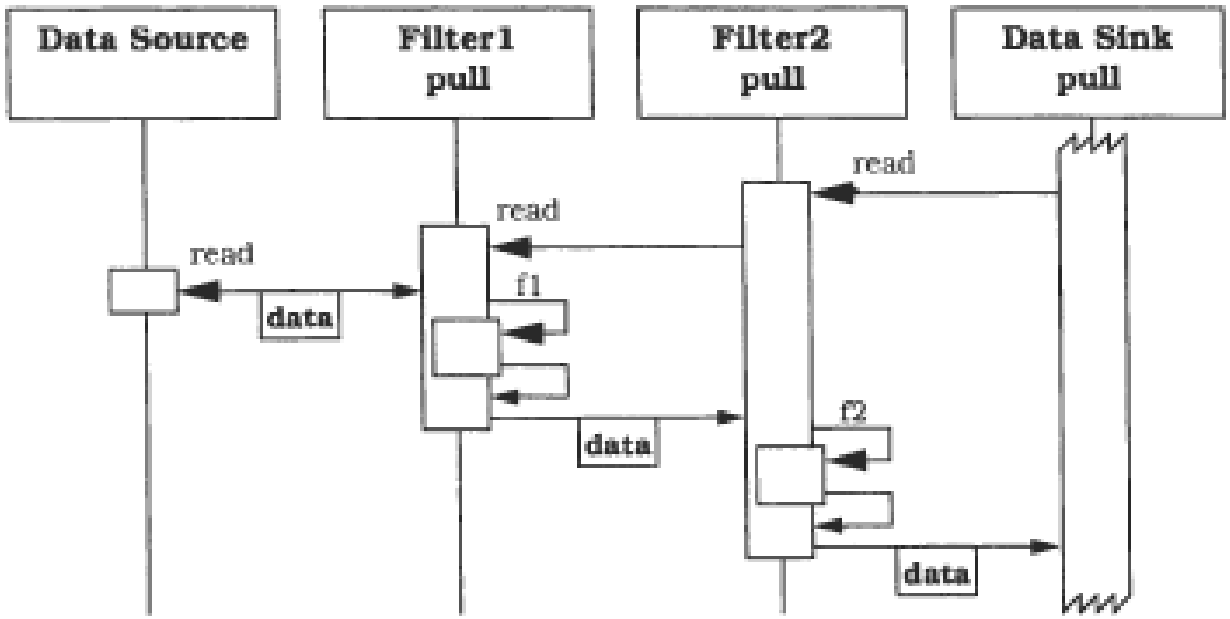

ativo
Parser
bool yyparse()
calls yylex
calls semantic checks
Scanner
token yylex()
calls getchar
Semantic Analyser
calls codegen
Input
int getchar()
Code Generator
writeCodeByte()
writes pipe
UNIX Pipe
read
write
reads pipe
AuLait Interpreter
Front End
Back End

- Dinâmica
  - Cenário 1



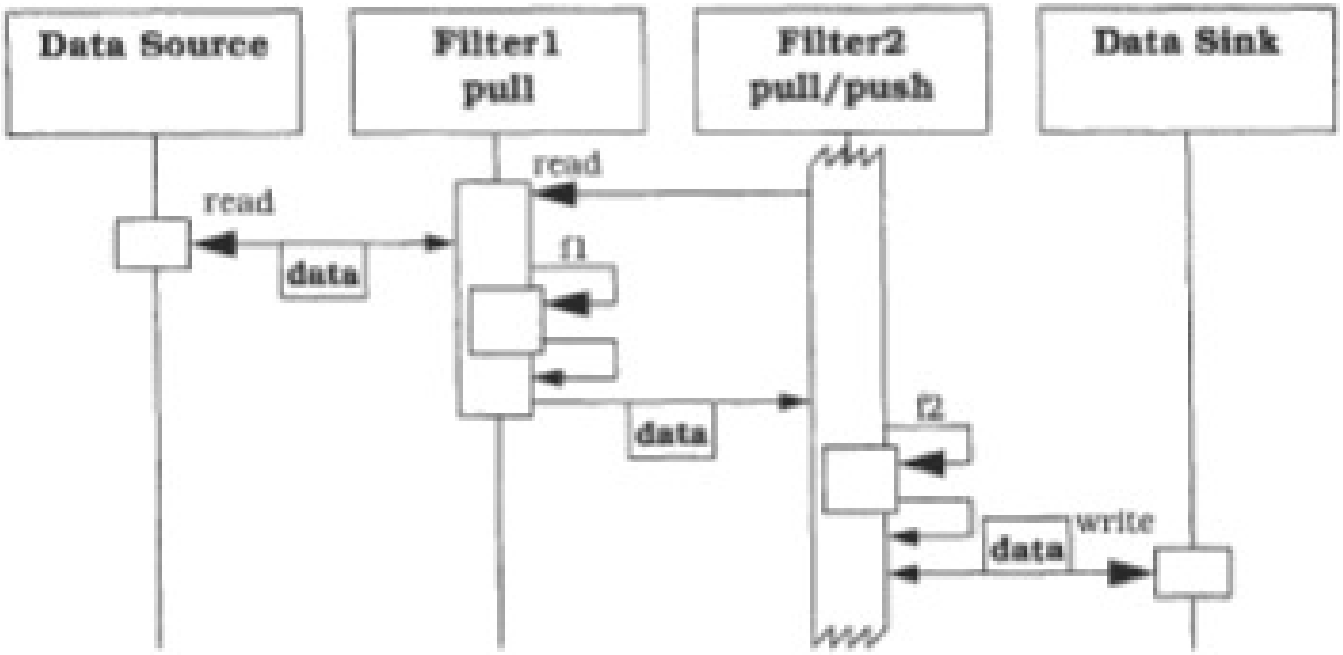
Filter 1 e Filter 2 são passivos (procedimentos)

- Cenário 2



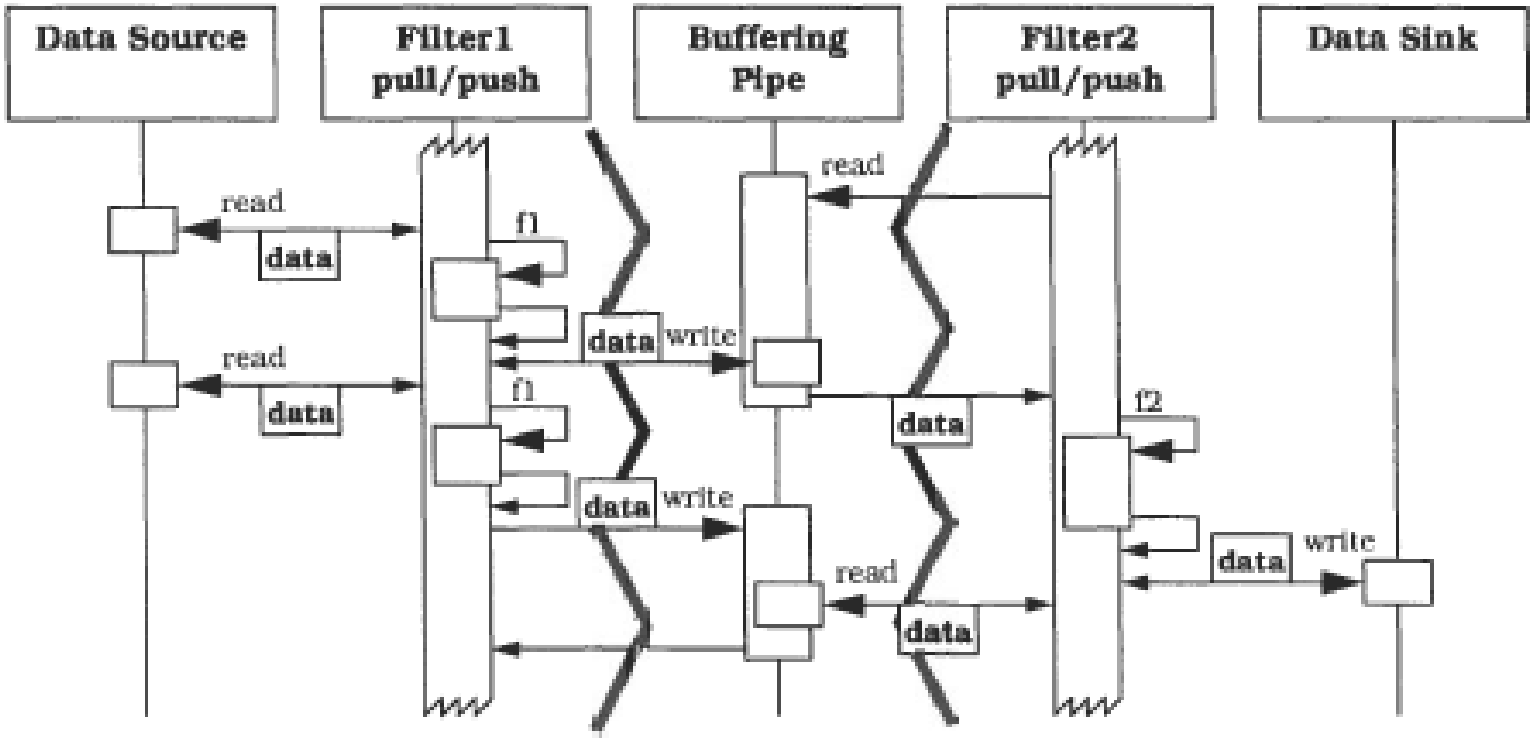
Filter 1 e Filter 2 são passivos (funções)

- Cenário 3

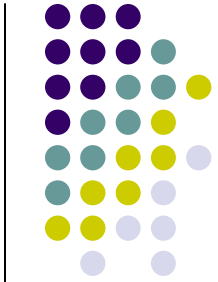


Filter 2 é ativo

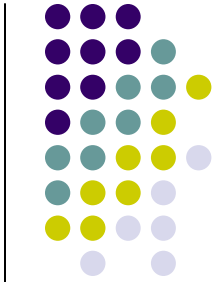
- Cenário 4



- O *pipe* é um buffer
- Se dados não estão disponíveis, os filtros são suspensos

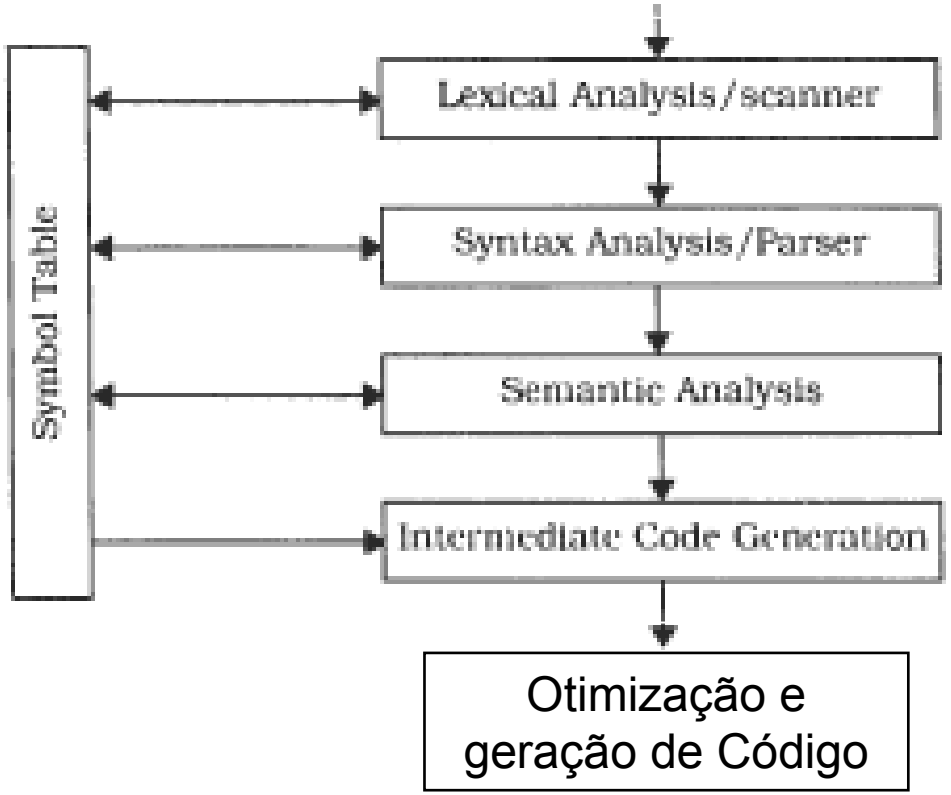


- Implementação
  - Direta usando chamadas a função ou procedimento, ou pipes de Unix
  - Numa linguagem funcional (Haskell, por exemplo) chamadas as funções são feitas por demanda

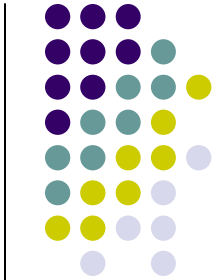


- Decisões
  1. Divida as tarefas do sistema em etapas de processamento seqüencial
  2. Defina os formatos dos dados a serem passados entre as etapas
  3. Decida como implementar cada conexão (pipe)
  4. Projete e implemente os filtros
  5. Projete tratamento de erros
     - Difícil e freqüentemente negligenciado
  6. Estabelecer o pipeline de processamento

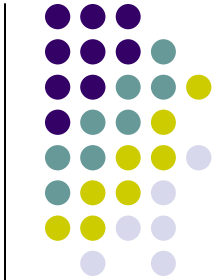
Exemplo



- Na prática, um compilador não segue o padrão estritamente (performance e compartilhamento de um estado global)

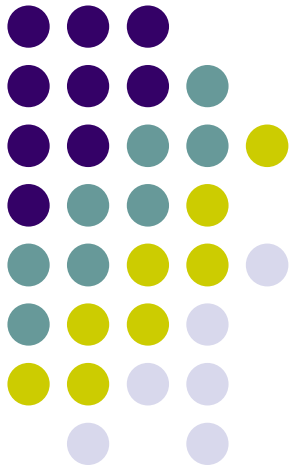


- Usos conhecidos
  - Pipes de Unix, CMS Pipelines (IBM), LASSPTools (análise numérica e gráficos)

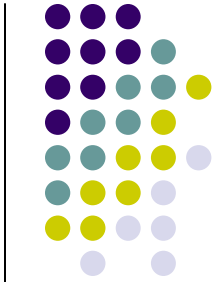


- Conseqüências
  - Arquivos intermediários não são necessários, mas é possível
  - Flexibilidade por troca de filtro
  - Flexibilidade por recombinação
  - Reuso de filtros
  - Prototipagem rápida de filtros
  - Eficiência por processamento paralelo
  - Dificuldades
    - Compartilhar informação
    - Transformação de dados produz sobrecarga
    - Tratamento de erros é difícil
- Ver também: padrão camadas

Padrões para Sistemas de Informação (Aplicações Corporativas)

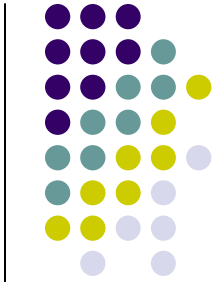


Conteúdo



- Aplicações Corporativas (AC)
- Arquitetura em Camadas
- Camadas típicas de AC
- Visão geral de alguns padrões para AC

Aplicações Corporativas



- Exemplos:
  - Folha de pagamentos, registro de pacientes, seguimento de vendas, análise de custos, ..
- Envolvem dados persistentes
- Grande massa de dados
- Acesso concorrente aos dados
- Muitas telas de interfaces com o usuário
- Integra com outras aplicações
- Dissonância conceitual com os dados
- “lógica” do negócio complexa

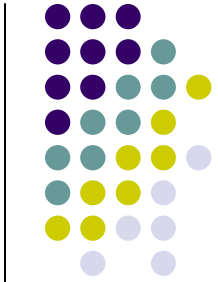
Divisão em Camadas

- Técnica comum para decompor um sistema complexo

Linguagem de Programação
Sistema Operacional
Drivers e CPU
Portas lógicas

FTP
TCP
IP
Ethernet
Conexão Física

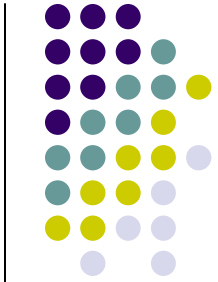
  - Camada superior só usa serviços de camada imediatamente inferior
  - Camada inferior desconhece camada superior
  - Na prática alguns sistemas relaxam estas regras
- POSA descreve camadas como um padrão arquitetural



- Benefícios
  - Entendimento de uma camada sem conhecer muito das outras
  - Pode substituir camadas com implementações alternativas dos serviços básicos
  - Minimiza dependências entre camadas
  - Permitem definição de standards
  - Uma camada pode ser usada para implementar vários serviços de mais alto nível

- Aspectos negativos
  - Nem tudo está bem encapsulado. Mudanças em cascata podem acontecer
  - Muitas camadas podem interferir com a performance

Camadas típicas das Aplicações Corporativas



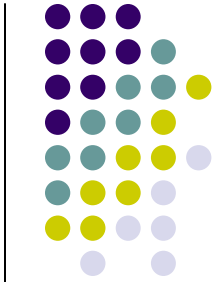
- Decidir que camadas ter e quais são as responsabilidades de cada camada é o mais difícil no projeto arquitetural
- As três principais camadas:

Camada	Responsabilidades
Apresentação	Prover serviços e apresentar informação
Lógica do Domínio	Lógica do negócio. A lógica fundamental do sistema
Fonte de Dados	Comunicação com BD, sistemas de mensagens e outros

Camada de Apresentação

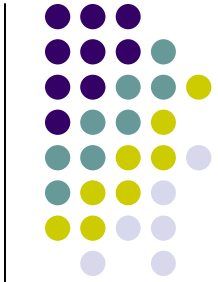
- Faz a interface entre o sistema e o usuário
  - Web, Swing, troca de arquivos, ...
- Recebe requisições do usuário, destina para objetos de negócio, recebe e formata a resposta

Camada do Domínio

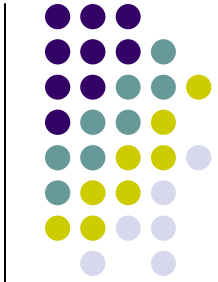


- É o coração do sistema
- Modelados os objetos do negócio que executam os processos solicitados
- Desejável que mapeiem objetos do mundo real
- Aqui é o único lugar onde as regras de negócio são executadas

Camada da Fonte de Dados



- Responsável por interagir com outros sistemas e por Persistir os dados
- Em geral, se comunica com um BD
- Age como um mapeador do mundo OO para o BD
- Serviço importante: interface para pesquisas



- Considerações
  - Camadas totalmente encapsuladas ou Apresentação acessa Dados diretamente
  - Domínio e Dados nunca devem depender da apresentação
  - Relação Domínio-Dados é mais complicada: depende dos padrões e ferramentas usadas
  - Múltiplos pacotes em cada camada
  - É difícil reconhecer o que é lógica do domínio

Outros esquemas de camadas

Brown	Fowler
Apresentação	Apresentação
Controlador / Mediador	Apresentação (Aplication Controller)
Domínio	Lógica do Domínio
Mapeamento de Dados	Fonte de Dados (Data Mapper)
Fonte de Dados	Fonte de Dados

Outros esquemas de camadas

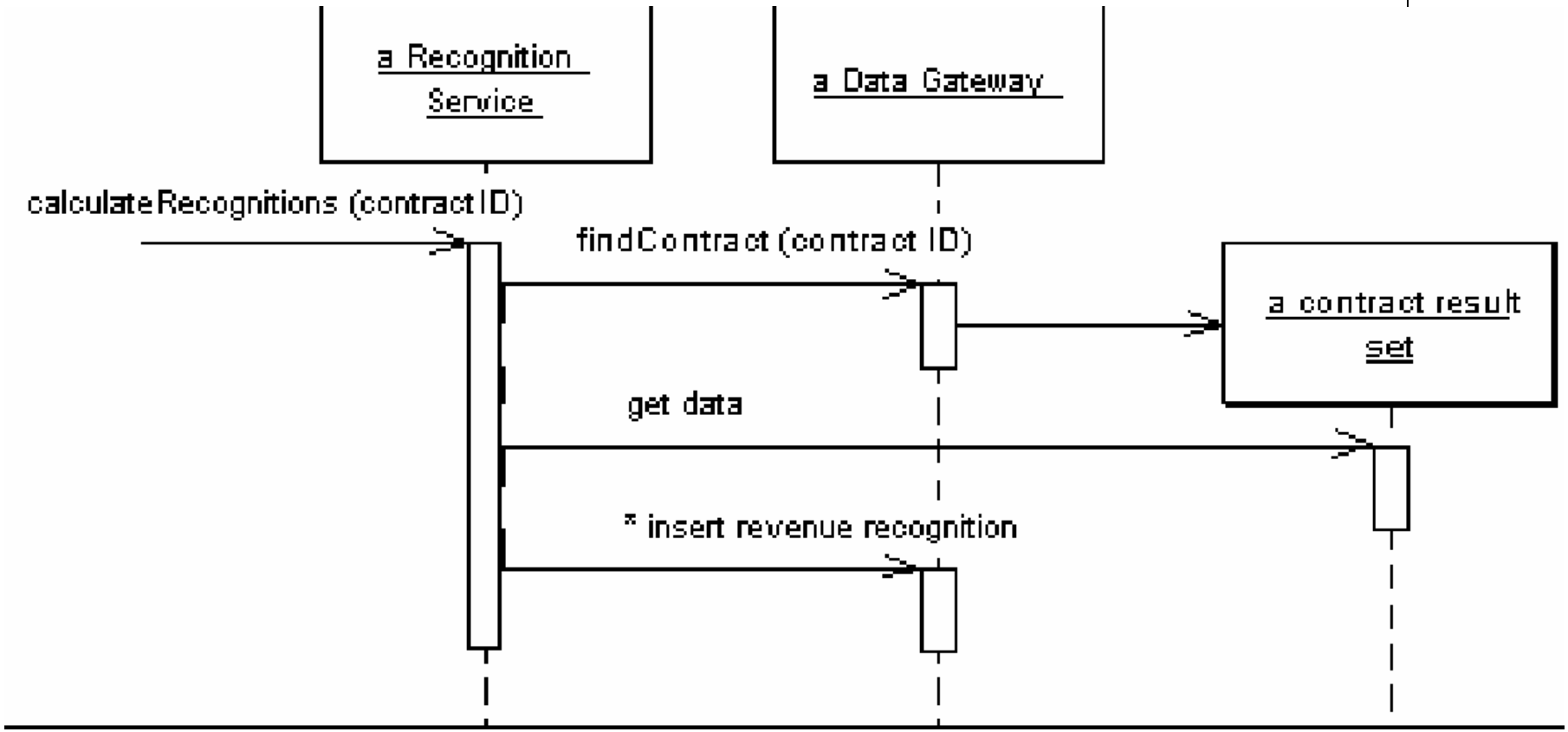
Core J2EE	Fowler
Cliente	Apresentação que roda no cliente (clientes ricos (swing ...))
Apresentação	Apresentação que roda no servidor (Server Pages, HTTP handlers, ...)
Negócio	Lógica do Domínio
Integração	Fonte de Dados
Recursos	Recursos externos (BD, outros sistemas, ...)

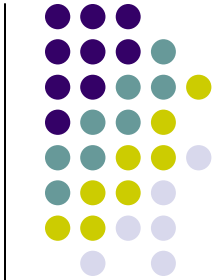
Outros esquemas de camadas

Marinescu	Fowler
Apresentação	Apresentação
Aplicação	Apresentação (Application Controller)
Serviços	Domínio (Service Layer)
Domínio	Domínio (Domain Model)
Persistência	Fonte de Dados

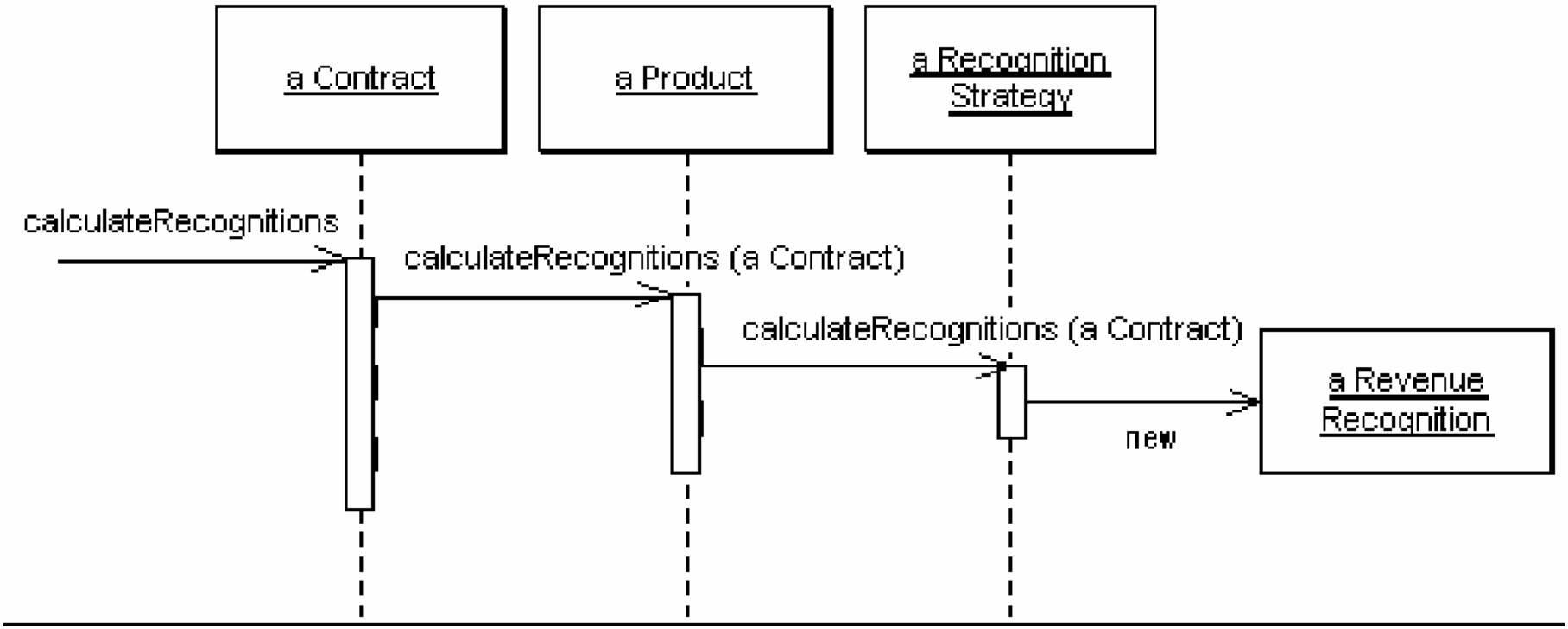
Padrões para a Lógica do Domínio

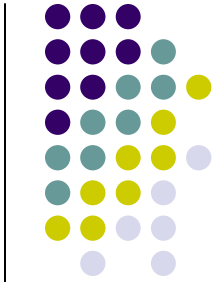
- *Transaction Script*
  - Padrão arquitetural útil em aplicações pequenas
  - Um procedimento para cada ação que o usuário pode desejar
  - Segue as regras da programação procedural
- Vantagens
  - Modelo simples que todo mundo entende
  - Trabalha bem com *Row Data Gateway* ou *Table Data Gateway*
- Desvantagens
  - Programação procedural, baixa reusabilidade, repetição de código, não extensível


a Recognition Service
a Data Gateway
calculateRecognitions (contractID)
findContract (contract ID)
a contract result set
get data
* insert revenue recognition

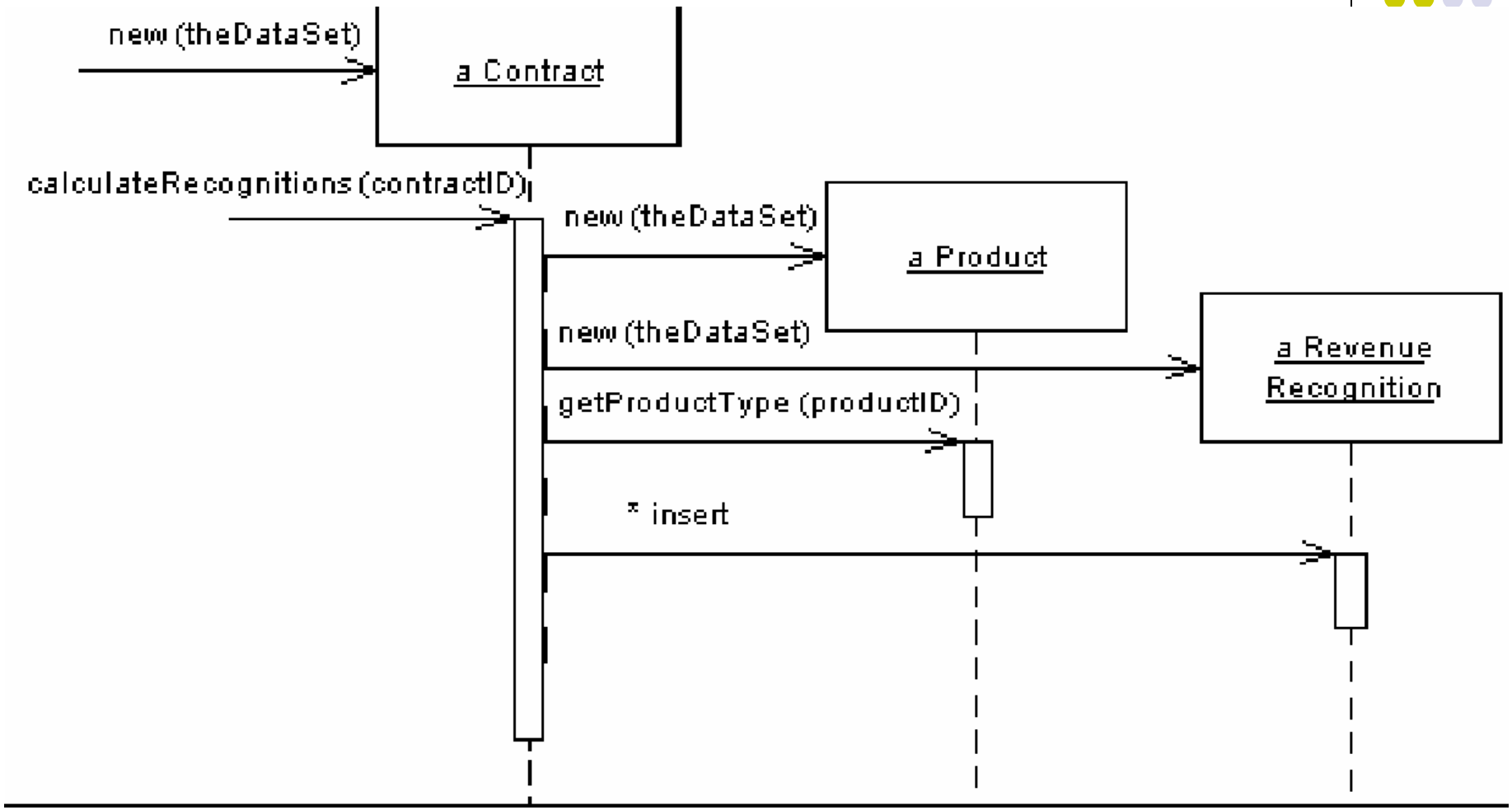


- *Domain Model*
  - Padrão arquitetural que usa o paradigma OO
- Vantagens
  - Usa e se beneficia de todos os princípios da OO
  - Eficaz para lógica complexa
  - Desenvolvimento por metáfora
  - Melhor evolução e manutenção
- Desvantagens
  - Complexo
  - Baixa produtividade
  - Difícil mapear a um BD

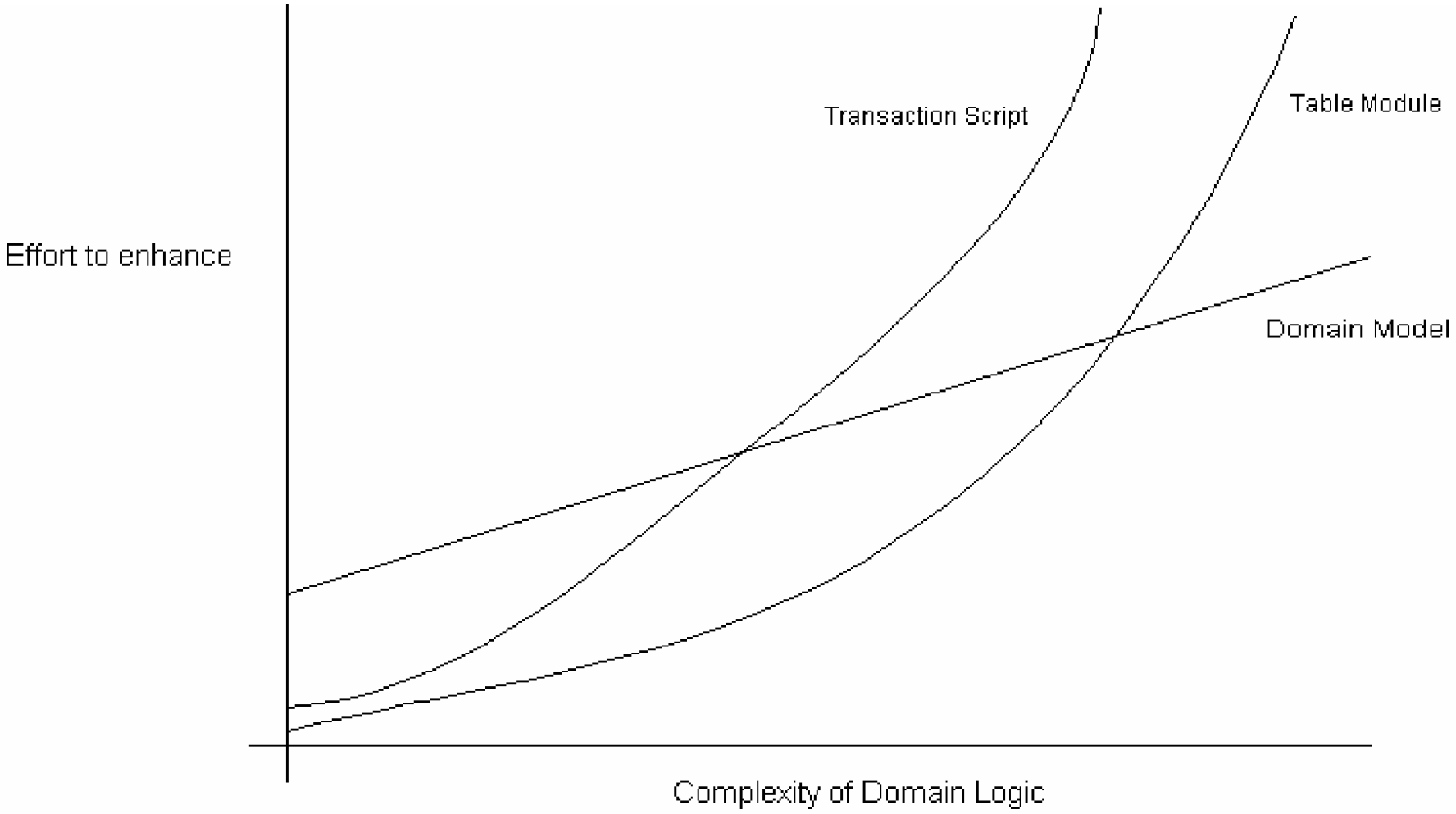

a Contract
a Product
a Recognition Strategy
calculateRecognitions
calculateRecognitions (a Contract)
calculateRecognitions (a Contract)
new
a Revenue Recognition



- *Table Module*
  - Organiza a lógica do negócio baseado em tabelas mais do que procedimentos
  - Uma instância representa todo um conjunto de registros do BD
  - Projetado para trabalhar com *Record Set*
- Vantagens
  - Cabe muito bem no resto da arquitetura
- Desvantagens
  - Não pode usar técnicas O.O.

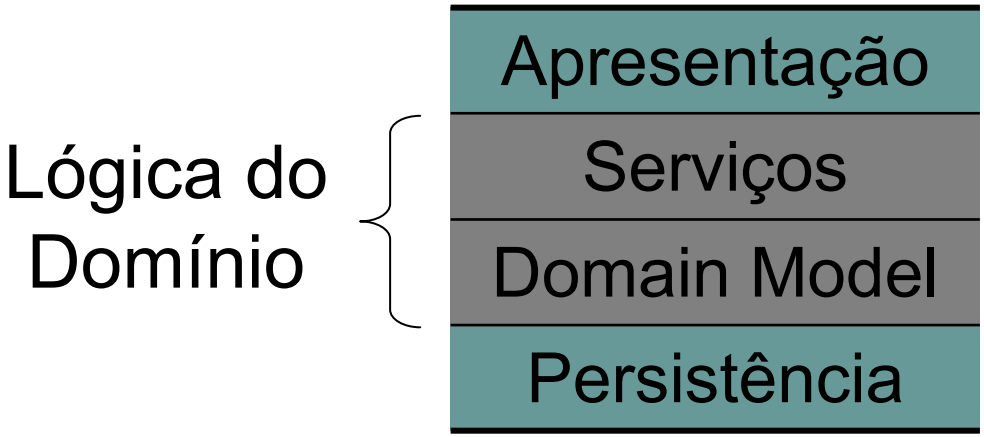

new (theDataSet)
a Contract
calculateRecognitions (contractID)
new (theDataSet)
a Product
new (theDataSet)
a Revenue Recognition
getProductType (productID)
* insert

Escolha do padrão para a lógica de domínio

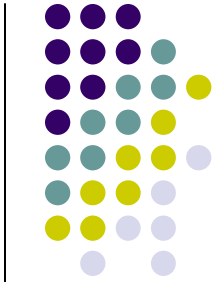


Camada de Serviço

- Abordagem comum: dividir a camada de Domínio em dois: Serviço e Domain Model (ou Table Model)

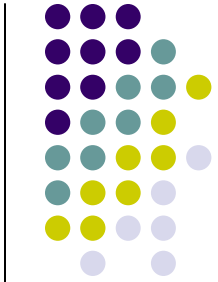


Camada de Serviço

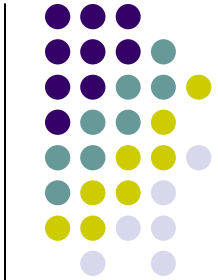


- Em geral:
  - Façade aos objetos do domínio, orientado pelos casos de uso
  - Controle de transação e segurança por método
  - Provê à Apresentação uma API fácil de usar
- Outra possibilidade
  - Maioria da lógica em Transaction Scripts usando simples objetos de domínio (Active Record, por exemplo)

Padrões da Camada de Serviço

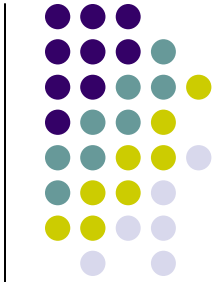


- Façade
  - Cria um ponto de entrada para cada processo
  - Exemplo de processo: adicionar um usuário
    - Criar objeto, preenchê-lo, localizar grupo, adicionar neste grupo, etc.
- Vantagens
  - Expõe conceitos, não detalhes de implementação
  - Controla bem casos de uso
  - Permite consistência e segurança (se a única entrada é a façade)
- Desvantagens
  - Limita a extensibilidade
  - Compromete reusabilidade

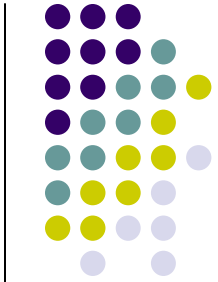


- Command
  - Encapsula um processo num objeto
  - Modelo seguido pelos frameworks MVC
- Vantagens
  - Fácil de entender
  - Operações Atômicas
- Desvantagens
  - Tende à programação procedural
  - Baixa reusabilidade
  - Lógica de negócio vaza para dentro do Command

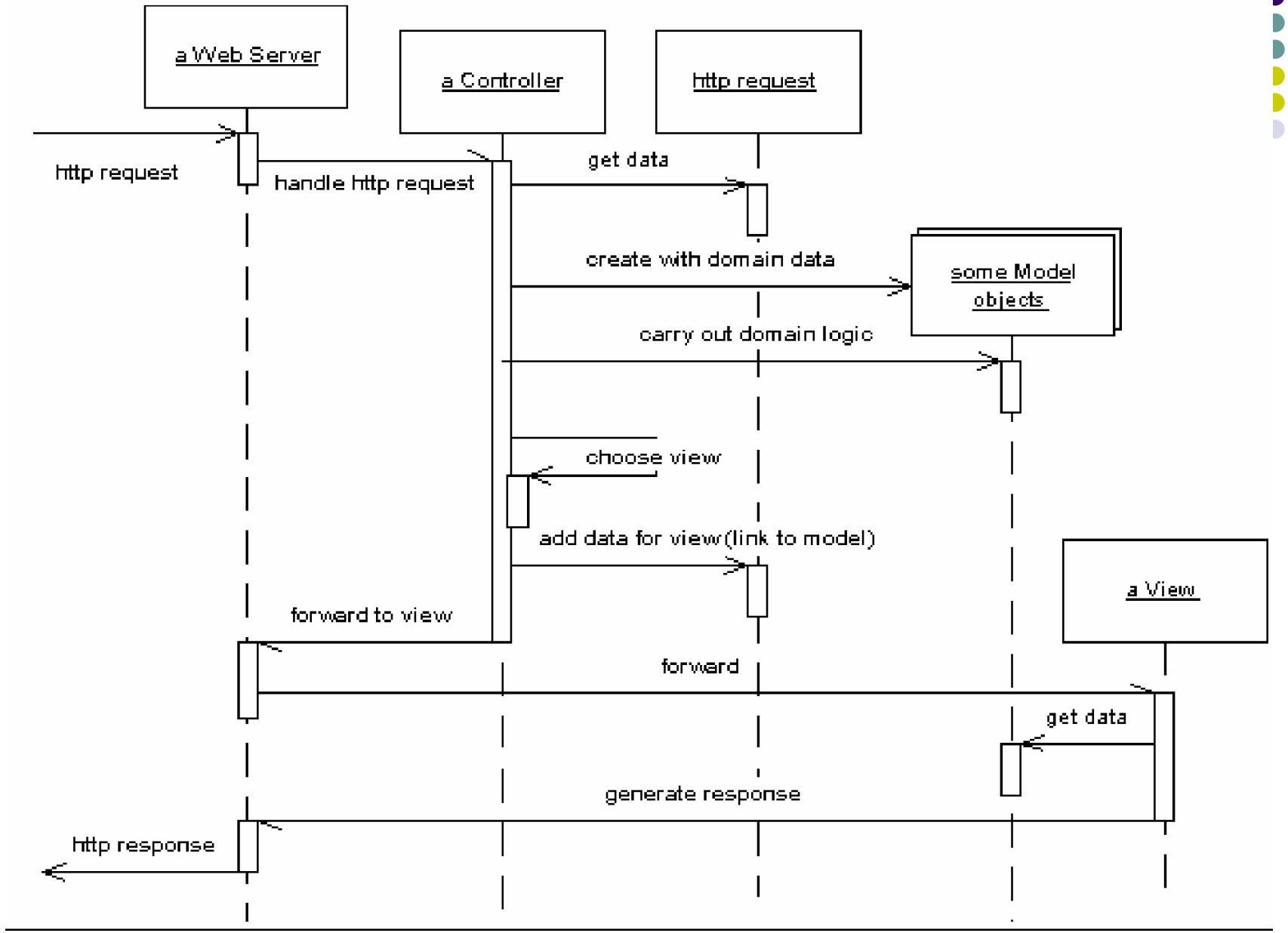
Padrões da Camada de Apresentação

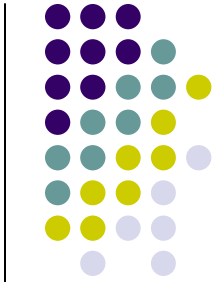


- Smart UI
  - Inclui lógica do domínio e persistência
  - Comumente em Visual Basic, ASP e PHP
- Vantagens
  - Produtividade alta
  - Não requer desenvolvedores experientes
  - Ferramentas RAD
- Desvantagens
  - Código repetido e desorganizado, alto acoplamento, baixa coesão, nenhuma separação de responsabilidades, não escala, difícil de manter
  - Torna-se um anti-padrão



- Model-View-Controller (MVC)
  - Três tipos de componentes
    - View: formulários ou páginas (JSP)
    - Controller: responde a solicitações e despacha para o Model
    - Model: (Interface para) Objetos do Negócio

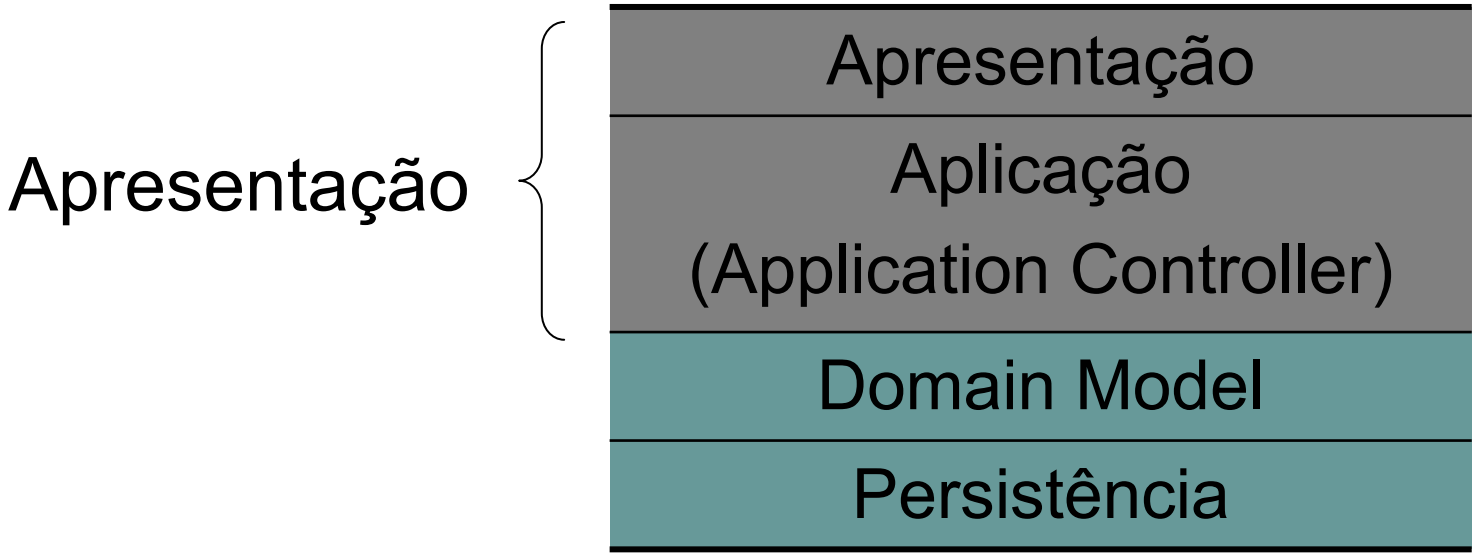

a Web Server
a Controller
http request
http request
handle http request
get data
create with domain data
some Model objects
carry out domain logic
choose view
add data for view(link to model)
a View
forward to view
forward
get data
generate response
http response



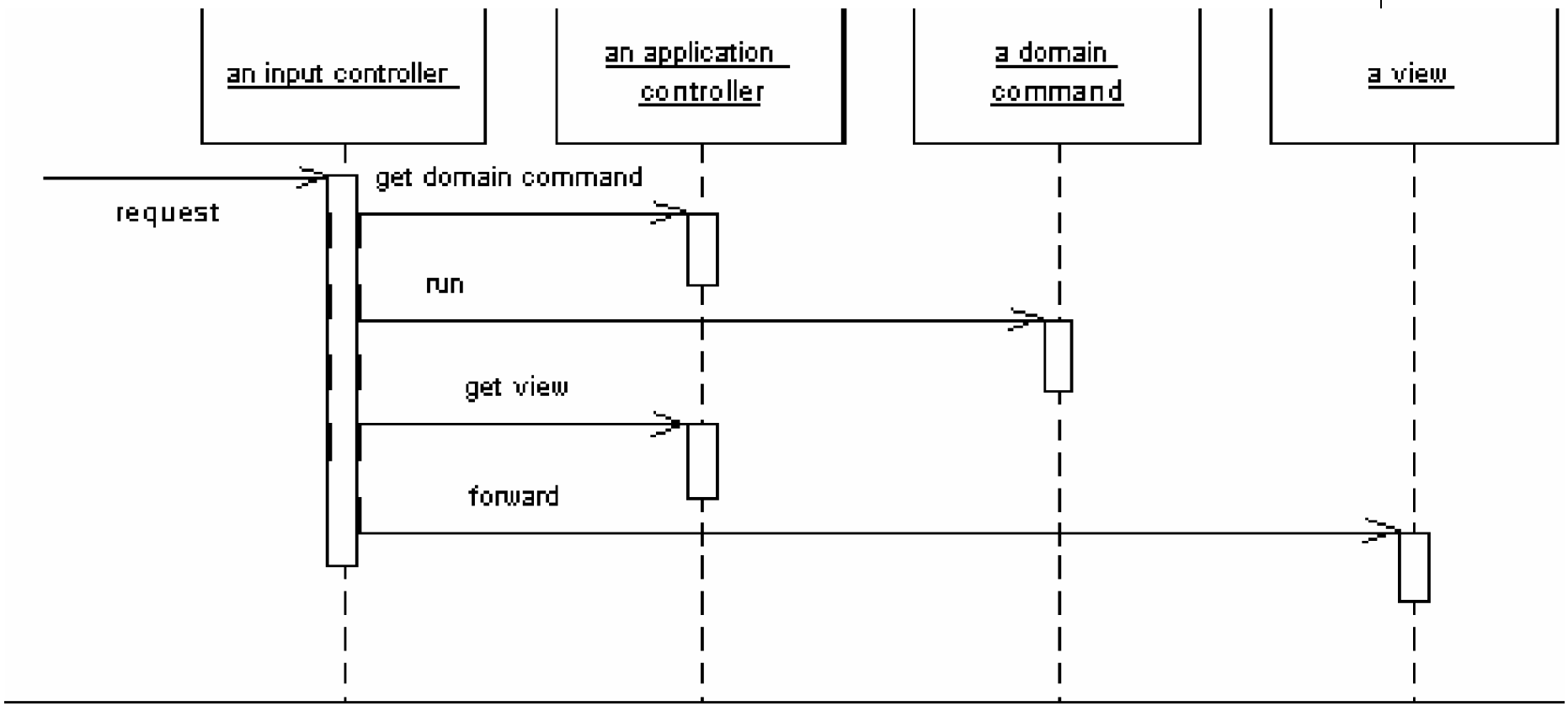
- Vantagens
  - Os modelos são completamente separados da Apresentação
  - Conhecido e documentado
  - Frameworks e Ferramentas disponíveis
- Desvantagens
  - Ideal para Desktop. Na Web, funciona com algumas limitações
  - Existem algumas interpretações erradas do padrão

Application Controler

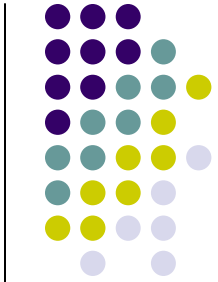
- Em alguns casos é útil dividir a Apresentação



- Application Controller
  - Manipula o fluxo da aplicação, decidindo que telas aparecem e em que ordem
  - Útil quando há uma lógica complexa de telas. Exemplo: sistemas tipo wizard
  - Independente da apresentação
- Vantagens
  - Evita duplicação de código nos controladores de entrada
- Desvantagem
  - Maior complexidade


an input controller
an application controller
a domain command
a view
request
get domain command
run
get view
forward

Padrões para a camada de Dados



- Data Mapper / Data Access Object (DAO)
  - Camada de mappers que move dados entre objetos e BDs, mantendo-os independentes um do outro
- Diferentes maneiras
  - Algumas suportadas por ferramentas como o Hibernate ou JDO