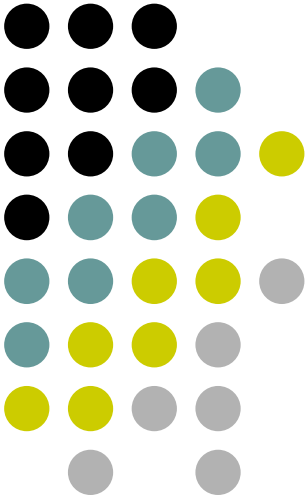
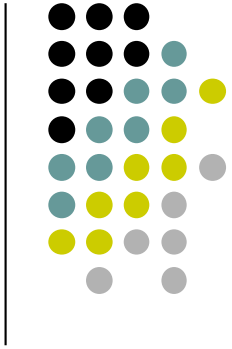
Sistemas Distribuídos

Módulo 2 – Escopo e Decisões de Projeto

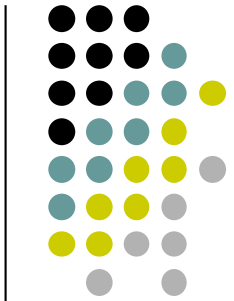


Aspectos a serem considerados



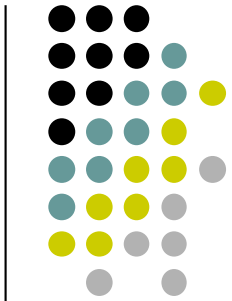
- Transparência
- Heterogeneidade
- Openess
- Segurança
- Tratamento de Falhas
- Concorrência
- Escalabilidade
- Desempenho

Transparência



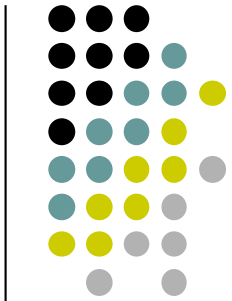
- **Objetivo:** fornecer aos usuários uma imagem única do sistema
  - Níveis de transparência:
    - Em nível de usuário: O usuário tem a impressão de manipular um sistema único.
    - Em nível de programador: O programador tem a ilusão de programar um sistema único.
      - Sintaxe e semântica das chamadas deve ser semelhante.

Tipos de Transparência (ISO RM-ODP)



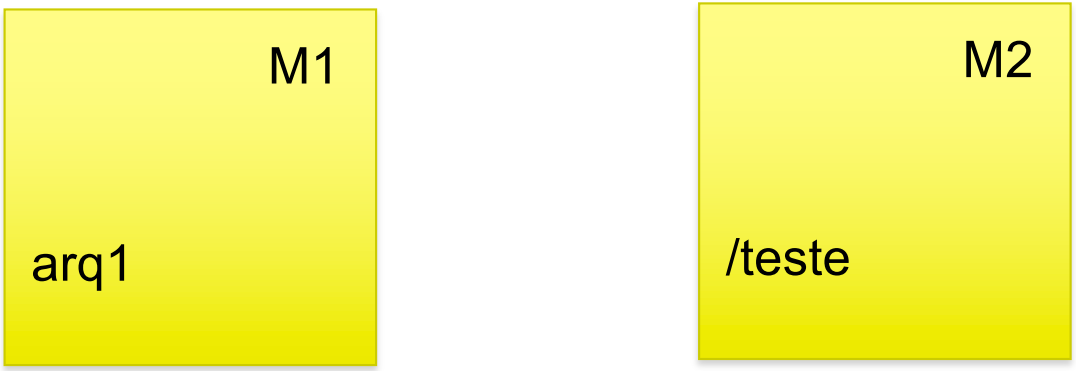
- Transparência de Acesso
- Transparência de Localização
- Transparência de Concorrência
- Transparência de Replicação
- Transparência de Falha
- Transparência de Migração
- Transparência de Paralelismo

Transparência de Acesso



- Transparência de Acesso e Localização são os tipos básicos de transparência
- Permite que recursos locais e remotos sejam acessados através de operações idênticas.
- Por exemplo, sistemas que só permitem o acesso a arquivos remotos via ftp não possuem transparência de acesso.
- O NFS provê transparência de acesso.

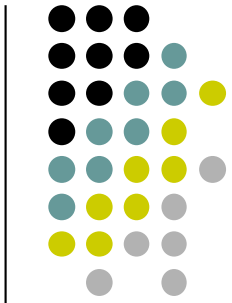
Exemplo de transparência de acesso



> cp arq1 /teste/. → transparente de acesso

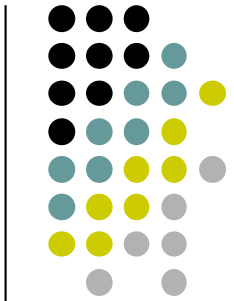
> sftp M2
connected to M2
sftp> cd /teste
sftp> put arq1 → não transparente de acesso

Transparência de Localização



- Os usuários não devem estar conscientes da localização física dos recursos.
  - Ex: o nome do recurso não deve conter o nome da máquina na qual o recurso reside (maq1:/usr/suporte/prog1.c).
- Os sistemas transparentes quanto à localização geralmente possuem um servidor de nomes, que fará o mapeamento entre o nome e o endereço.

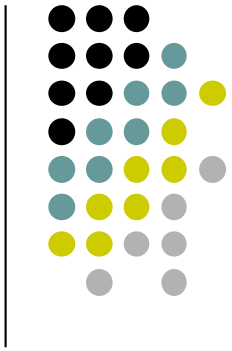
Transparência de Concorrência



- Os usuários não devem notar que existem outros usuários no sistema. Se dois usuários acessam simultaneamente um mesmo recurso, o sistema deve garantir a coerência.
- UTOPIA => Nem os sistemas em uma única máquina garantem esta condição.

☛ *Em sistemas distribuídos, devem ser garantidas as mesmas condições de concorrência de uma única máquina*

Transparência de Concorrência



- Concorrência em máquina única:

processo 1	processo 2
a=1; b=1;	c=b; d=a;

Execução 2:

a=1; ⟶	b=1; ⟶	c=1; ⟶	d=1; ⟶	c=d=1;
c=0; ⟶	a=1; ⟶	b=1; ⟶	d=1; ⟶	c=0,d=1;

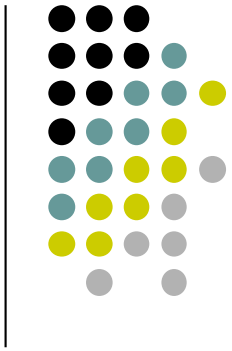
Máquina única

a = 0
b = 0
c = 0
d = 0

processo 1	processo 2
`a=1;`	`c=b;`
`b=1;`	`d=a;`

Outputs possíveis:
P1 -> P2: c=1, d=1
P2 -> P1: c=0, d=0
P1 -> P2 -> P1: c=0, d=1
P2 -> P1 -> P2: c=0, d=1

Transparência de Concorrência



- Concorrência em sistemas distribuídos:

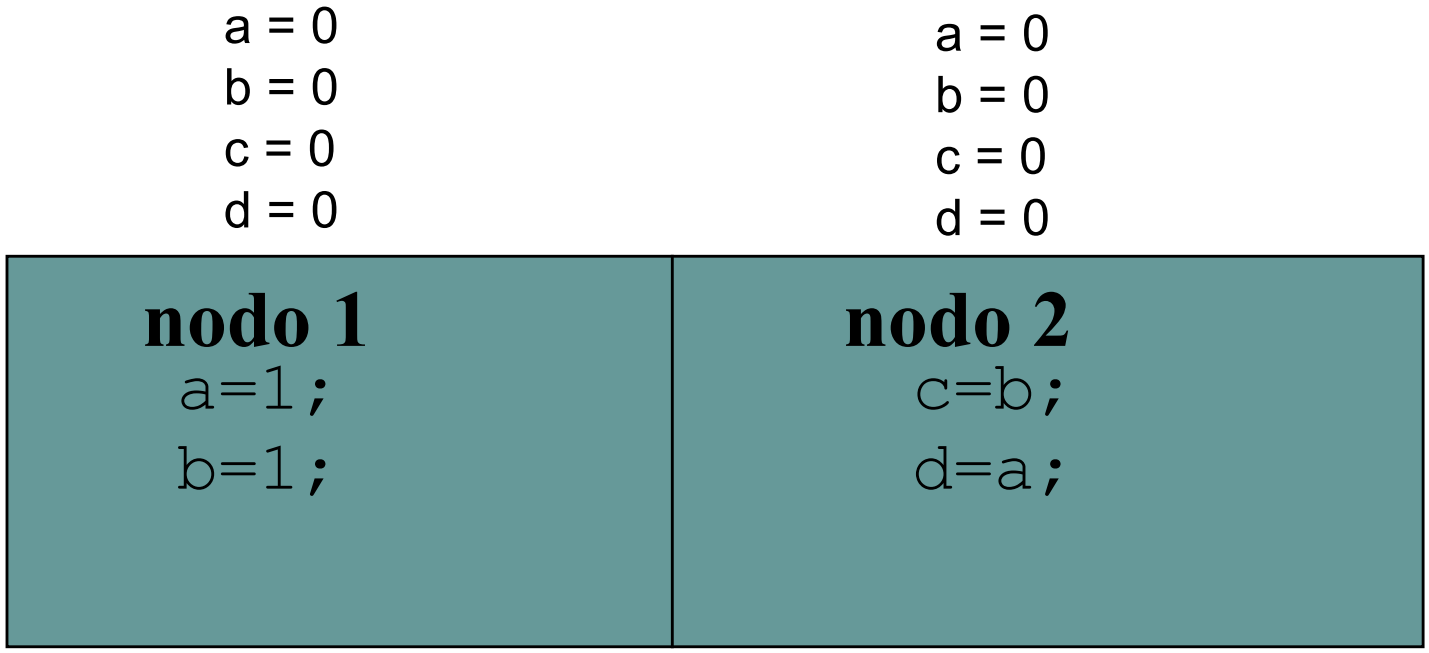
nodo 1	**nodo 2**
`a=1;`	`c=b;`
`b=1;`	`d=a;`

Execução 1:

`b=1;` ⟶ `c=1;` ⟶ `d=0;` ⟶ `a=1;` ⟶ `c=1,d=0;`

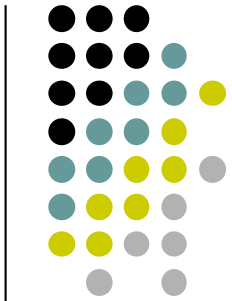
Violação da ordem do programa !!!

Duas Máquinas



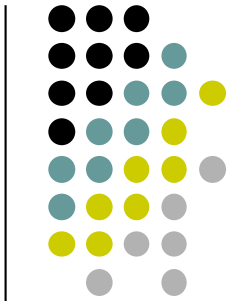
Outputs possíveis:
P1 -> P2 pode ser c=1, d=1
P1 -> P2 pode ser c=1, d=0

Transparência de Replicação



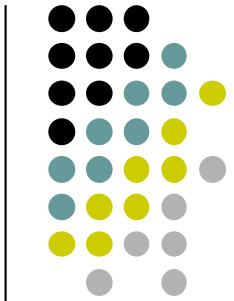
- Permite que diversas cópias do mesmo recurso sejam utilizadas sem que os usuários ou programadores estejam conscientes de sua existência.
- O sistema garante, então, a coerência entre as diversas cópias.

Transparência de Falha



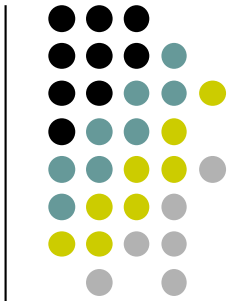
- Permite que as tarefas sejam completadas, mesmo na ocorrência de falhas.
- Um sistema deste tipo deve ter um mecanismo de detecção e recuperação de falhas

Transparência de Migração



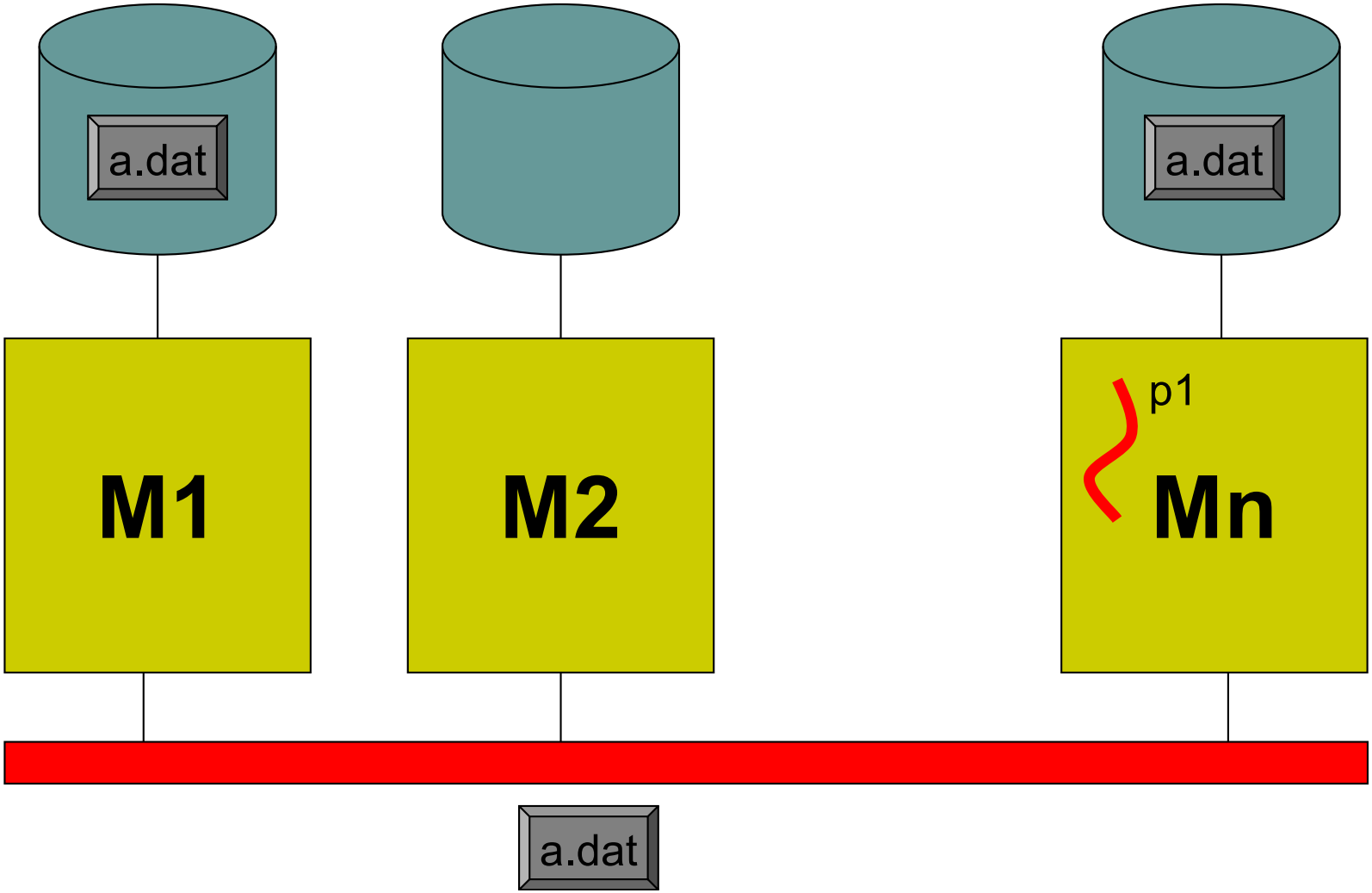
- Os recursos podem trocar de lugar à vontade no sistema. Um sistema transparente quanto à migração é necessariamente transparente quanto à localização, mas, além disso, deve observar outras características de projeto.
- O que pode migrar?
  - ✓ Dados
  - ✓ Computação
  - ✓ Processos

Transparência de Migração

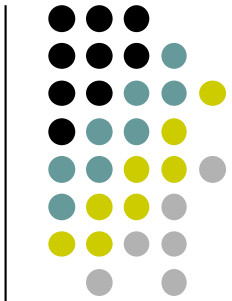


- Migração de dados
  - *Transferência de arquivos*: Quando um usuário necessita acessar um arquivo x, o arquivo x completo é transferido para a sua máquina local. Se houver alterações, o arquivo deve ser transferido de volta ao site origem
  - *Transferência de partes do arquivo*: Somente as partes do arquivo que serão acessadas são realmente transferidas.

Transparência de Migração
Migração de Dados

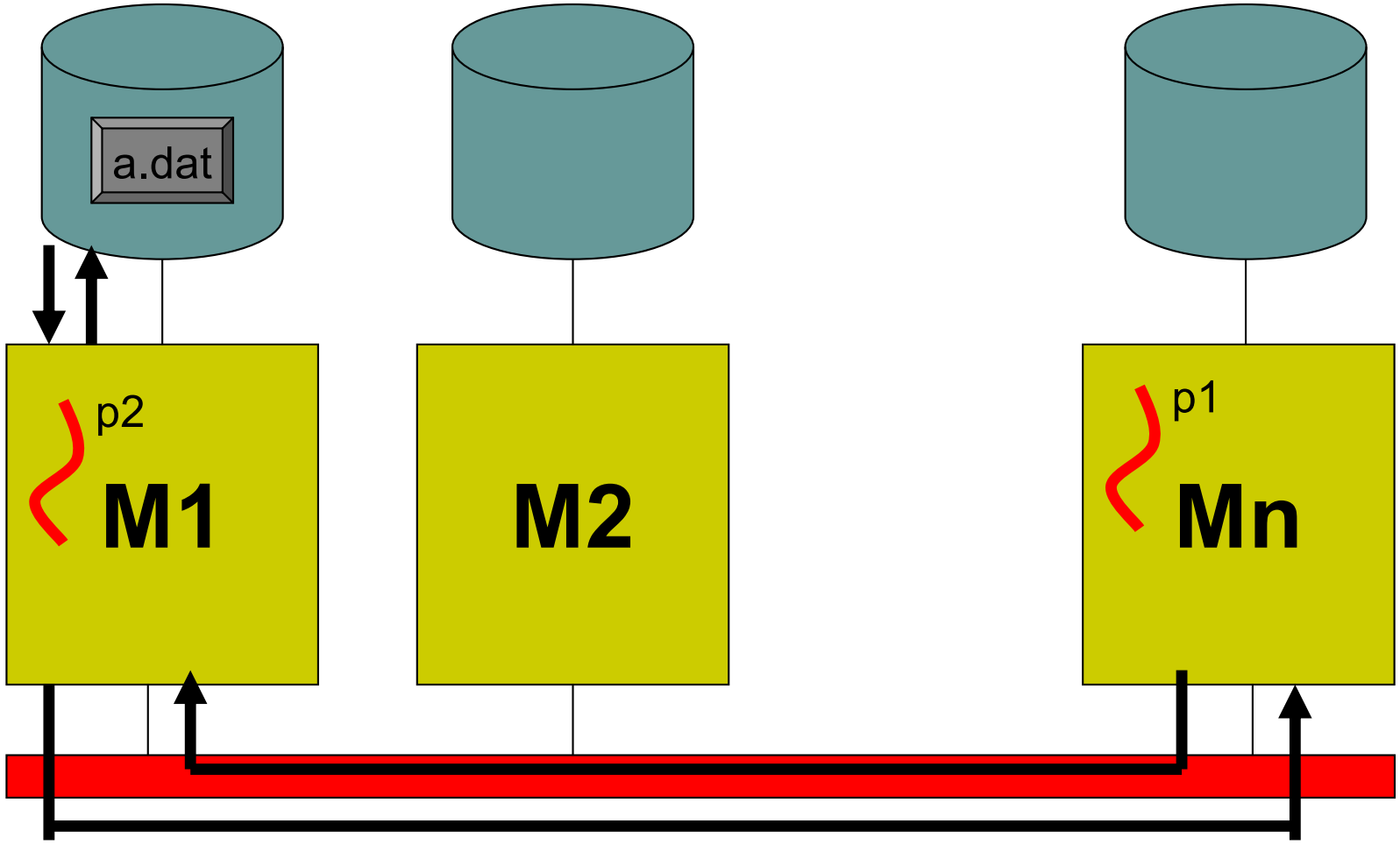


Transparência de Migração

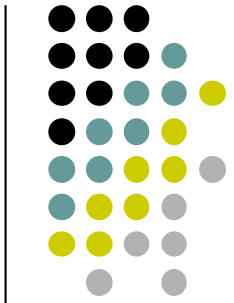


- Migração de computação
  - Quando se necessita de um grande volume de dados que se encontra em outro nodo, é mais eficiente transferir a computação do que transferir os dados.
  - A migração de computação pode ser feita via RPC ou pelo envio de mensagens

Transparência de Migração
Migração de Computação

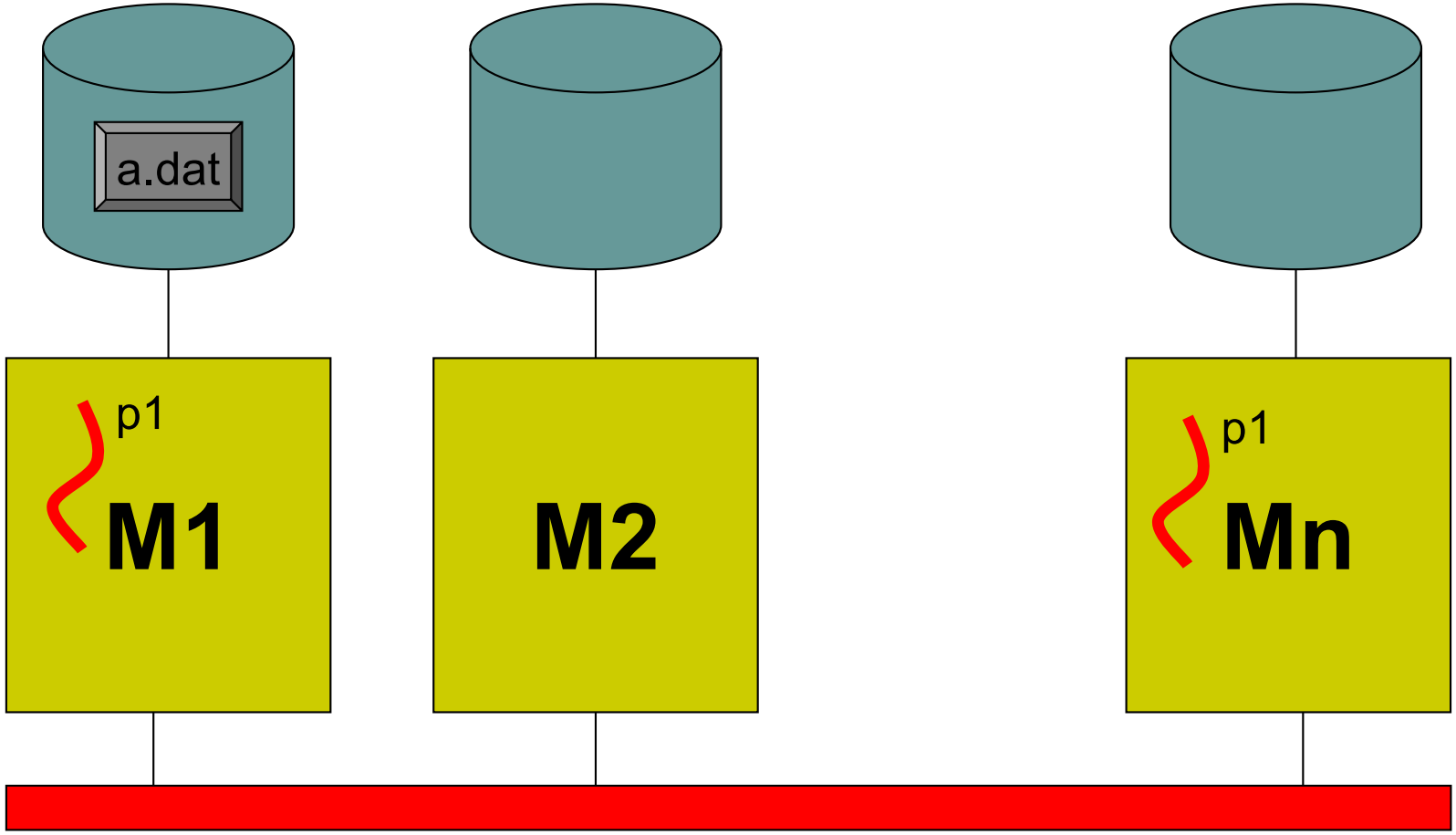


Transparência de Migração

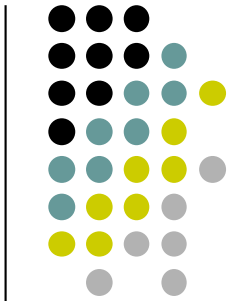


- Migração de processos
  - A migração de um processo, depois de iniciada a sua execução, pode ser justificada pelas seguintes razões:
    - Balanceamento de carga
    - Crash de um nodo
    - Preferências de hardware
    - Preferências de software

Transparência de Migração
Migração de Processos

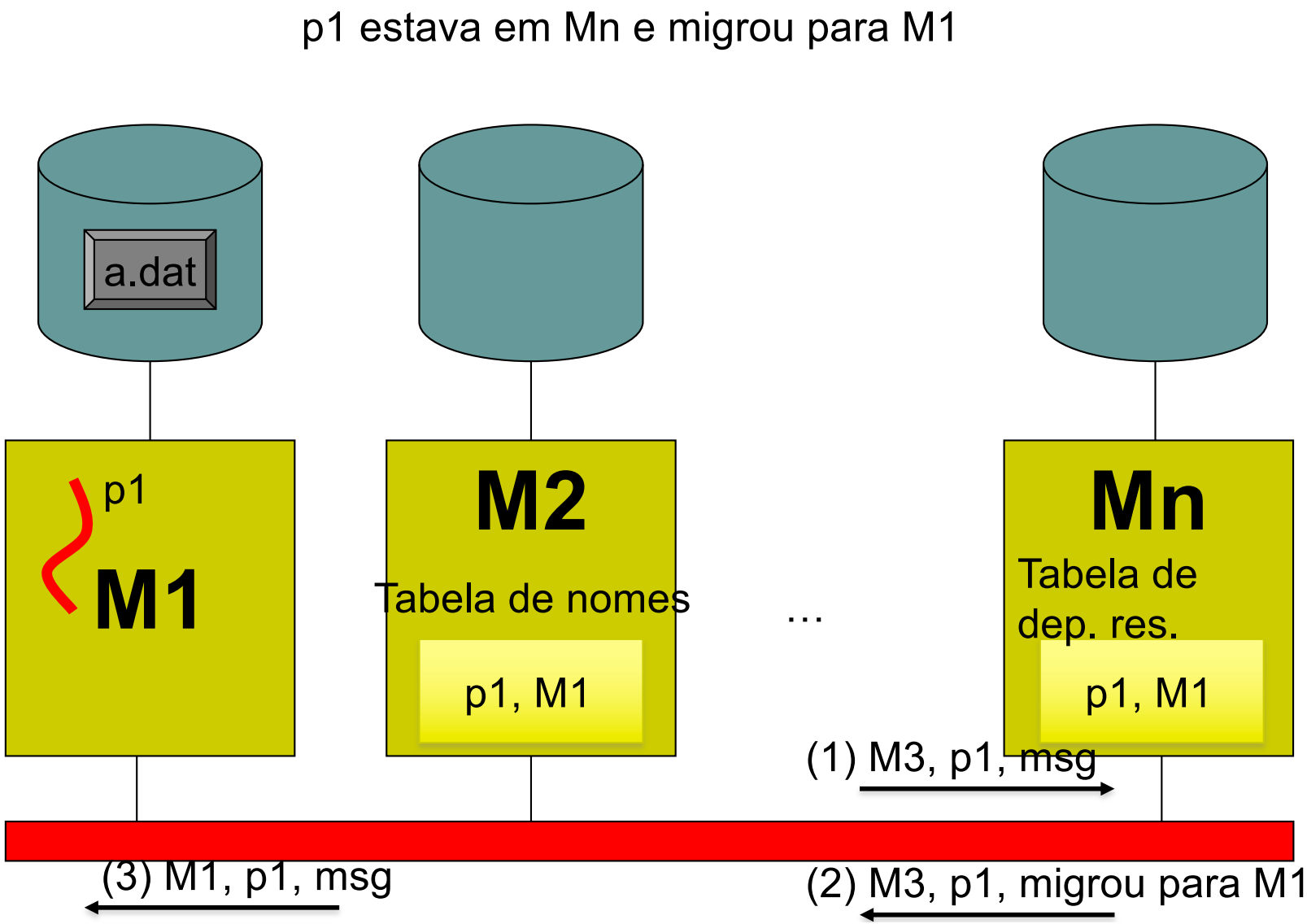


Transparência de Migração

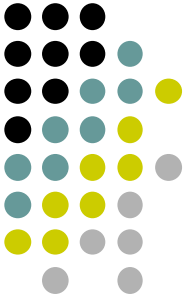


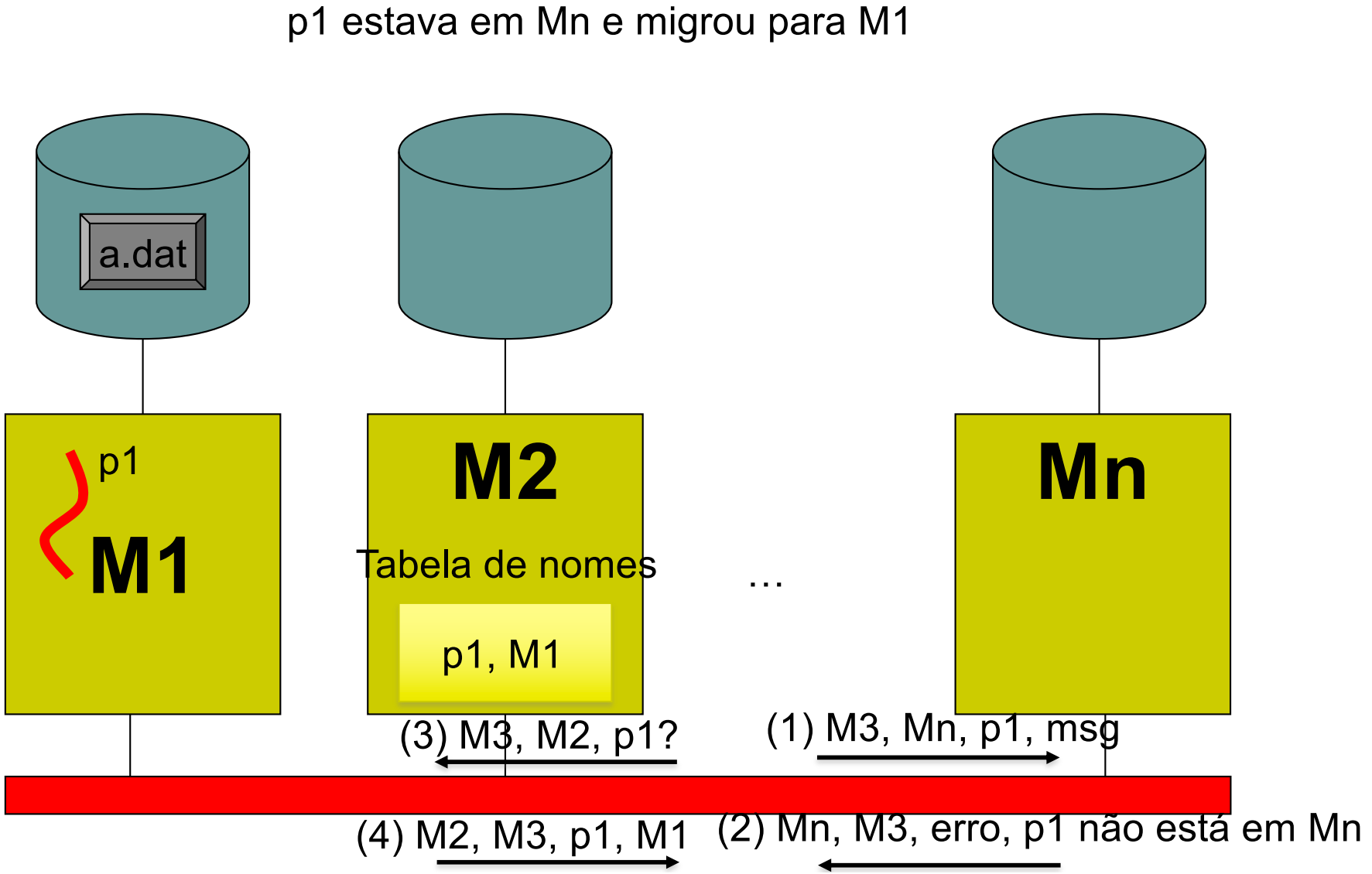
- Quando um recurso migra, podem haver solicitações em andamento no sistema para este componente, que não tomaram ainda conhecimento da nova localização.
- Em um sistema com dependência residual, os nodos guardam um histórico do movimento dos recursos, para que o processo que possua sua localização antiga (nome antigo) possa localizá-lo.
- Em sistemas sem dependência residual, a requisição falha e o nodo origem da solicitação, sabendo que o sistema permite migração, solicita a nova localização ao sistema.

Transparência de migração – Com dependência residual

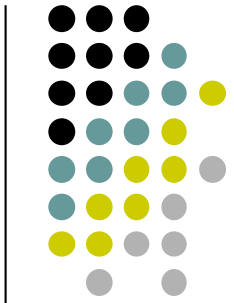


Transparência de migração – Sem dependência residual



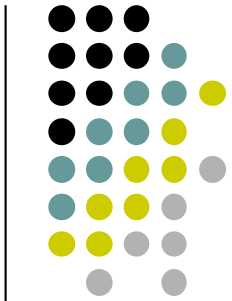


Transparência de Paralelismo



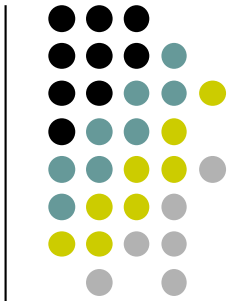
- O próprio sistema decide que recursos (e.g processadores) alocar a uma aplicação distribuída de maneira que critérios de otimização sejam atendidos (balanceamento de carga, tempo de resposta, etc).
- O usuário não deve interferir nesta escolha.
- Logo, dependendo da carga do sistema, o número de recursos alocados a uma aplicação pode variar de uma execução para outra.

Heterogeneidade



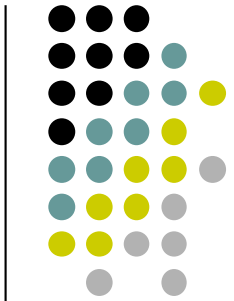
- Em um sistema distribuído, podemos ter os seguintes tipos de heterogeneidade:
  - Rede de interconexão
  - Hardware
  - Sistema operacional
  - Linguagem de programação
- Os diversos tipos de redes de interconexão que compõem o sistema geralmente são mascarados pelo uso do mesmo protocolo.

Heterogeneidade



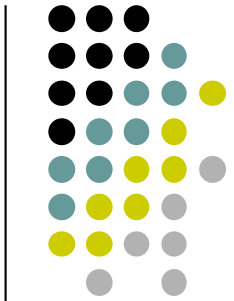
- Os diferentes hardwares de computadores são mascarados (de maneira restrita) quando é definida uma representação padrão de dados para troca de mensagens.
- Os diferentes sistemas operacionais são mascarados pelo uso de interfaces de comunicação de alto nível

Heterogeneidade



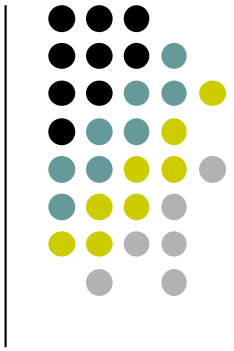
- Middleware: é uma camada de software (software layer) que provê um modelo computacional uniforme, mascarando a heterogeneidade da plataforma de hardware e software.
  - Exemplos de middleware: CORBA, Java RMI, Web Services

Openess



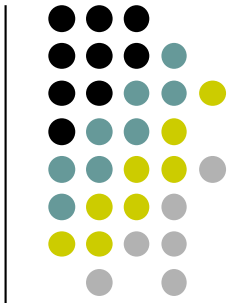
- É determinada pelo grau no qual o sistema pode ser estendido.
- É alcançada pela documentação e publicação das interfaces de software dos componentes, sendo o primeiro passo no sentido da padronização.
- Geralmente, as interfaces são publicadas através de RFCs (Requests for Comments), introduzidas pelos projetistas dos protocolos da Internet.

Segurança



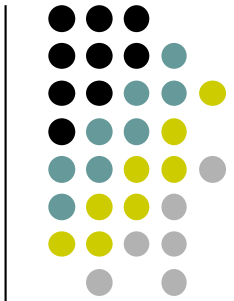
- A segurança em sistemas distribuídos envolve 3 componentes:
  - Confidencialidade: proteção contra indivíduos não autorizados
  - Integridade: proteção contra alterações e corrupção da informação
  - Disponibilidade: proteção contra a indisponibilidade da informação

Tratamento de Falhas



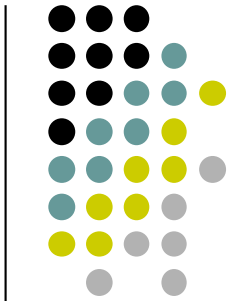
- Com a eliminação de uma singularidade (o processador), o sistema é potencialmente mais confiável.
- Na teoria, em um sistema de n processadores, a probabilidade de todos falharem ao mesmo tempo é $(1\text{-conf})^n$, onde n é o número de processadores.
- Na prática, a falha de um componente que seja afeta bastante o sistema e, em vários casos, o sistema como um todo cai.

Tratamento de Falhas



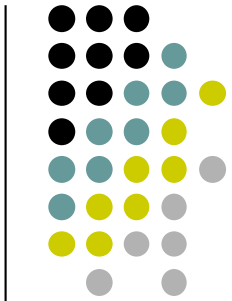
- O tratamento de falhas geralmente envolve as seguintes técnicas:
  - Detecção
  - Mascaramento
  - Tolerância
  - Recuperação após falhas
  - Redundância

Concorrência



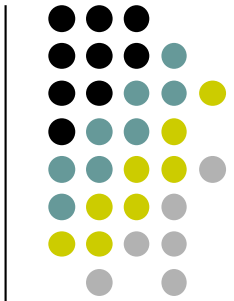
- O sistema distribuído oferece recursos que são compartilhados entre diversas aplicações.
- Existe a possibilidade de tentativa de acessos ao mesmo recurso ao mesmo tempo.
- Mecanismos de sincronização devem ser utilizados.

Escalabilidade



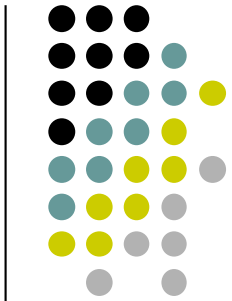
- Noção intuitiva: um sistema que funciona bem com 200 máquinas deve funcionar bem também com 200.000 máquinas.
- O desempenho da máquina não deve ser degradado a medida que o número de elementos (processadores, usuários) cresce de maneira significativa.

Escalabilidade



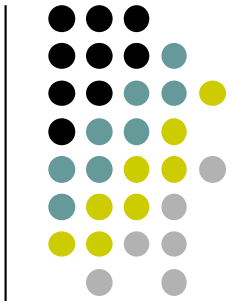
- Na realidade, nós temos vários níveis de escalabilidade:
  - arquitetura,
  - sistema operacional,
  - linguagem de programação,
  - aplicação distribuída

Escalabilidade



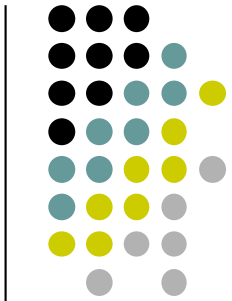
- Para se obter escalabilidade, o projeto do sistema deve manter fixo o número de recursos gerenciados por uma entidade e, à medida que o número de recursos cresce, aumentar também o número de gerentes.
- Estruturas hieráquicas de dados e de controle devem ser utilizadas
- Algoritmos devem ser descentralizados, de modo a evitar gargalos

Desempenho



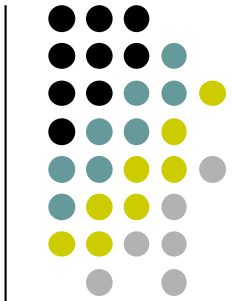
- Justificativa: já que temos mais máquinas, é de se esperar que obtenhamos melhor desempenho.
- Um dos maiores limitadores da performance de sistemas distribuídos é o *custo de comunicação*.
- Custo de comunicação: custo do protocolo + custo da transmissão física da mensagem.

Desempenho



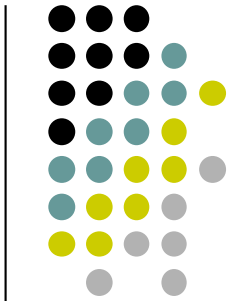
- Para se obter performance, deve-se então reduzir a comunicação entre os processadores.
- O problema principal é que, reduzindo a comunicação, podemos estar reduzindo o paralelismo e, consequentemente, reduzindo a performance.
- Devemos, então, achar um compromisso entre a comunicação e o paralelismo

Desempenho



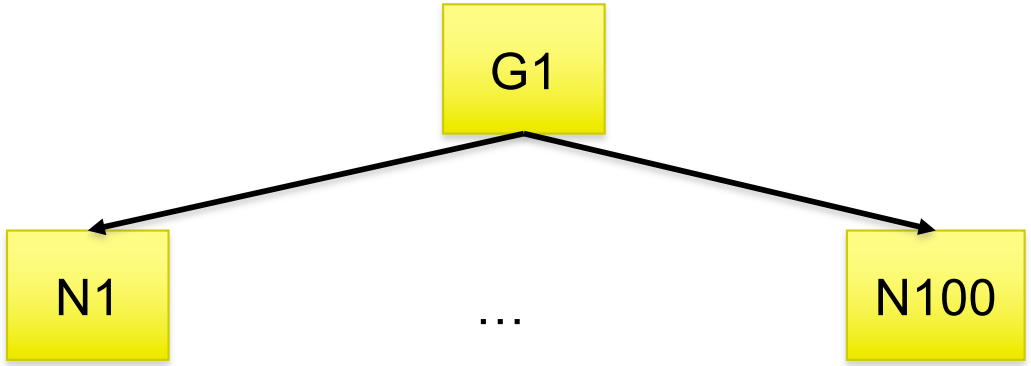
- Granulosidade: determina o elemento básico que será executado em paralelo.
  - ✓ fina: as instruções ou um conjunto pequeno de instruções são executados em paralelo
  - ✓ média: as funções são executadas em paralelo (a nível de threads).
  - ✓ grossa: processos são executados em paralelo (grande quantidade de código para cada processo).

Desempenho

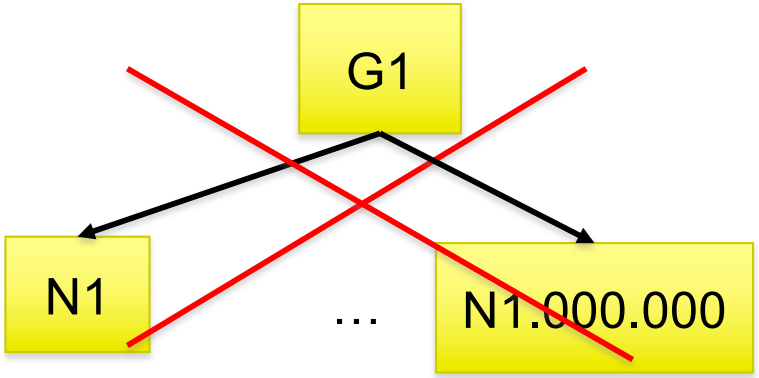


- Influência da granulosidade no desempenho de um sistema distribuído:
  - ✓ granulosidade fina => alta comunicação => performance reduzida
  - ✓ granulosidade grossa => pouca comunicação => performance melhora
  - ✓ Granulosidade a nível de aplicação => sistema monoprocessado.

Um gerente e 100 nodos



Como gerenciar 1 milhão de nodos?



Cada gerente gerencia um número limitado de nodos