ACH 2147 — DESENVOLVIMENTO DE SISTEMAS DE INFORMAÇÃO DISTRIBUÍDOS

ARQUITETURAS

Daniel Cordeiro

21 e 23 de março de 2018

Escola de Artes, Ciências e Humanidades | EACH | USP

ARQUITETURAS DECENTRALIZADAS

P2P estruturado os nós são organizados seguindo uma estrutura de dados distribuída específica

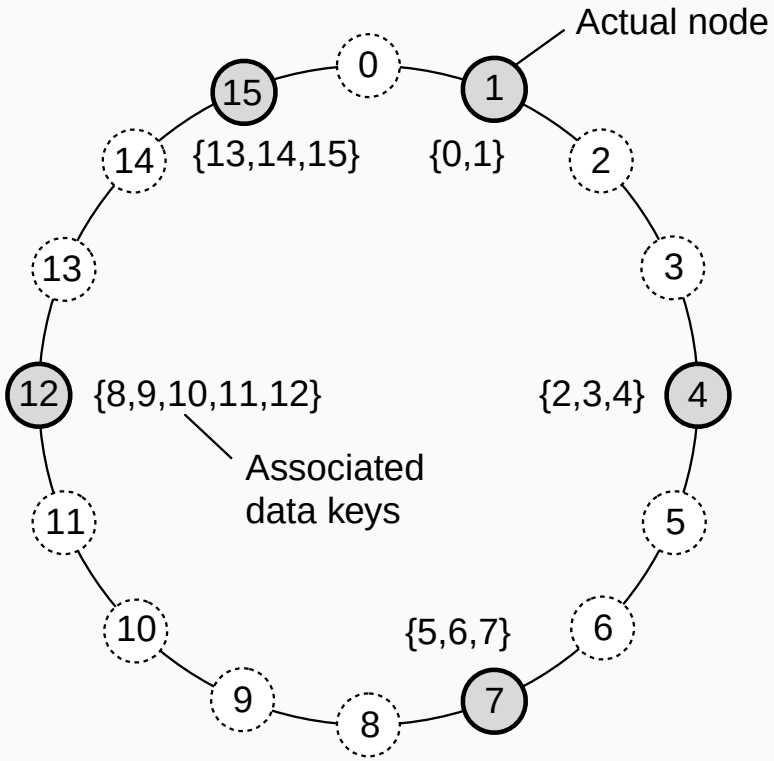
P2P não-estruturado os nós selecionam aleatoriamente seus vizinhos

P2P híbrido alguns nós são designados, de forma organizada, a executar funções especiais

Nota:
Praticamente todos os casos são exemplos de redes overlay: dados são roteados usando conexões definidas pelos nós (*Cf.* multicast em nível de aplicação)

Ideia básica

Organizar os nós em uma rede overlay estruturada, tal como um anel lógico, e fazer com que alguns nós se tornem responsáveis por alguns serviços baseado unicamente em seus IDs.



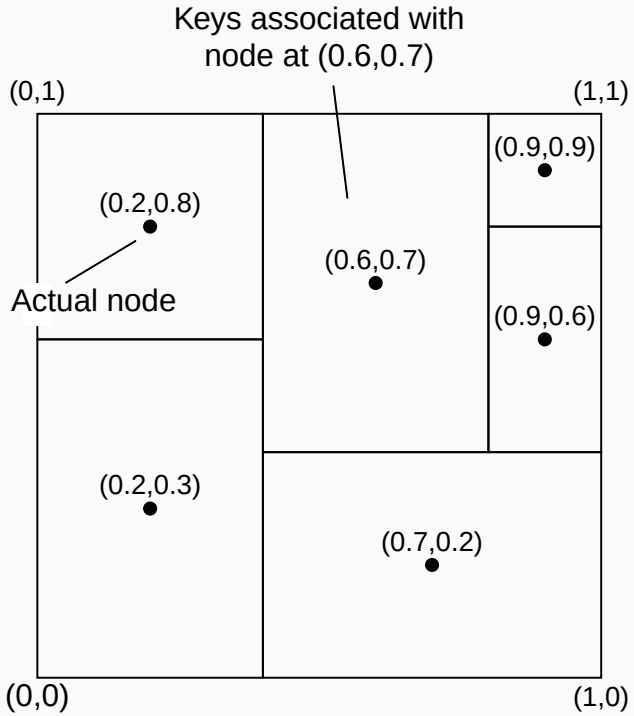
Nota

O sistema provê uma operação `LOOKUP(key)` que irá fazer o roteamento de uma requisição até o nó correspondente.

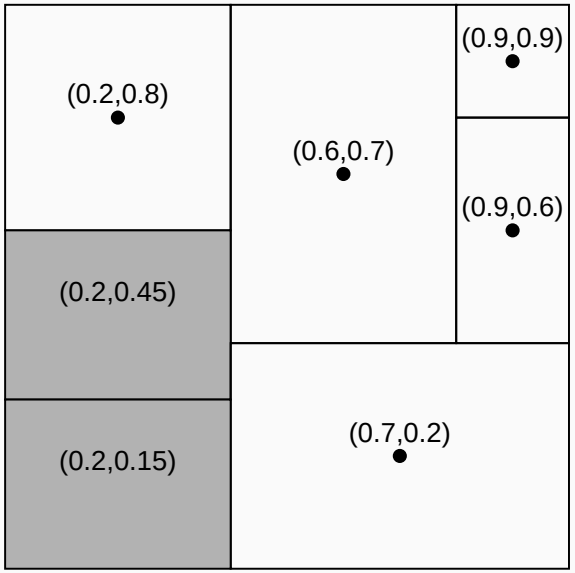
Outro exemplo

Organize os nós em um espaço d-dimensional e faça todos os nós ficarem responsáveis por um dado em uma região específica.

Quando um nó for adicionado, divida a região.



(a)



(b)

Observação
Muitos sistemas P2P não-estruturados tentam manter um grafo aleatório.

Princípio básico
Cada nó deve contactar um outro nó selecionado aleatoriamente:

- Cada participante mantém uma visão parcial da rede, consistindo de c outros nós
- Cada nó P seleciona periodicamente um nó Q de sua visão parcial
- P e Q trocam informação && trocam membros de suas respectivas visões parciais

Nota
Dependendo de como as trocas são feitas, não só a aleatoriedade mas também a robustez da rede pode ser garantida.

O QUE É GOSSIPING?

Thread ativa

Thread passiva

O QUE É GOSSIPING?

Thread ativa

```
selectPeer(&B);
```

Thread passiva

selectPeer: Seleciona aleatoriamente um vizinho de sua visão parcial.

O QUE É GOSSIPING?

Thread ativa

```
selectPeer(&B);
selectToSend(&bufs);
```

Thread passiva

selectPeer:	Seleciona aleatoriamente um vizinho de sua visão parcial.
selectToSend:	Seleciona s entradas de seu cache local.

O QUE É GOSSIPING?

Thread ativa

```
selectPeer(&B);
selectToSend(&bufs);
sendTo(B, bufs);
```

Thread passiva

```
receiveFromAny(&A, &bufr);
```

selectPeer:	Seleciona aleatoriamente um vizinho de sua visão parcial.
selectToSend:	Seleciona s entradas de seu cache local.

O QUE É GOSSIPING?

Thread ativa

```
selectPeer(&B);
selectToSend(&bufs);
sendTo(B, bufs);
```

Thread passiva

```
receiveFromAny(&A, &bufr);
selectToSend(&bufs);
```

selectPeer:	Seleciona aleatoriamente um vizinho de sua visão parcial.
selectToSend:	Seleciona s entradas de seu cache local.

O QUE É GOSSIPING?

Thread ativa

```
selectPeer(&B);
selectToSend(&bufs);
sendTo(B, bufs);

receiveFrom(B, &bufr);
```

Thread passiva

```
receiveFromAny(&A, &bufr);
selectToSend(&bufs);
sendTo(A, bufs);
```

selectPeer:	Seleciona aleatoriamente um vizinho de sua visão parcial.
selectToSend:	Seleciona s entradas de seu cache local.

O QUE É GOSSIPING?

Thread ativa

```
selectPeer(&B);
selectToSend(&bufs);
sendTo(B, bufs);

receiveFrom(B, &bufr);
selectToKeep(cache, bufr);
```

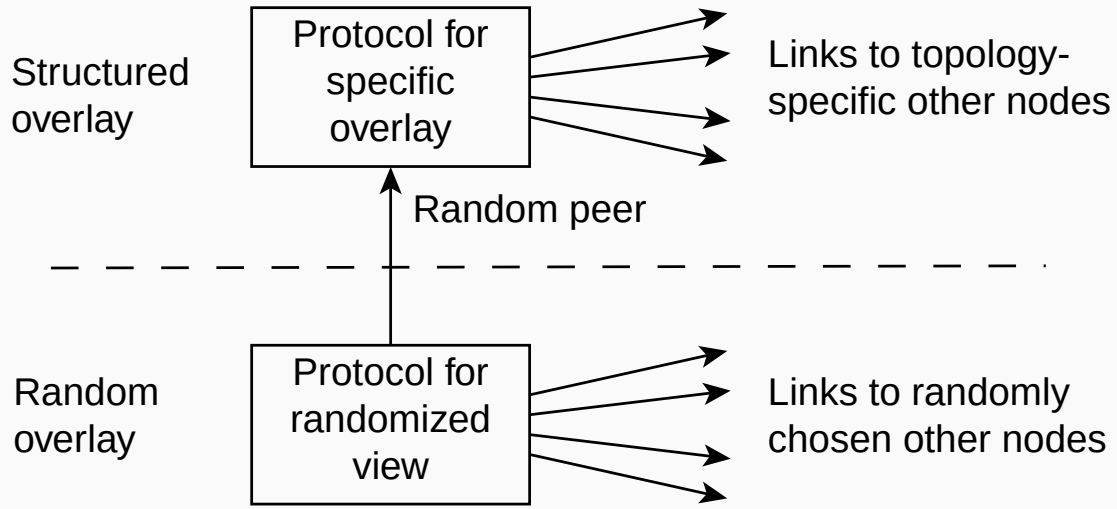
Thread passiva

```
receiveFromAny(&A, &bufr);
selectToSend(&bufs);
sendTo(A, bufs);
selectToKeep(cache, bufr);
```

selectPeer:	Seleciona aleatoriamente um vizinho de sua visão parcial.
selectToSend:	Seleciona s entradas de seu cache local.
selectToKeep:	(1) Adiciona as entradas recebidas ao cache local. (2) Remove os itens repetidos. (3) Encolhe o tamanho do cache para c (usando alguma estratégia).

Ideia básica

Distinguir duas camadas: (1) mantém uma visão parcial aleatória na camada inferior; (2) seleciona quem manter nas visões parciais das camadas superiores.



Nota

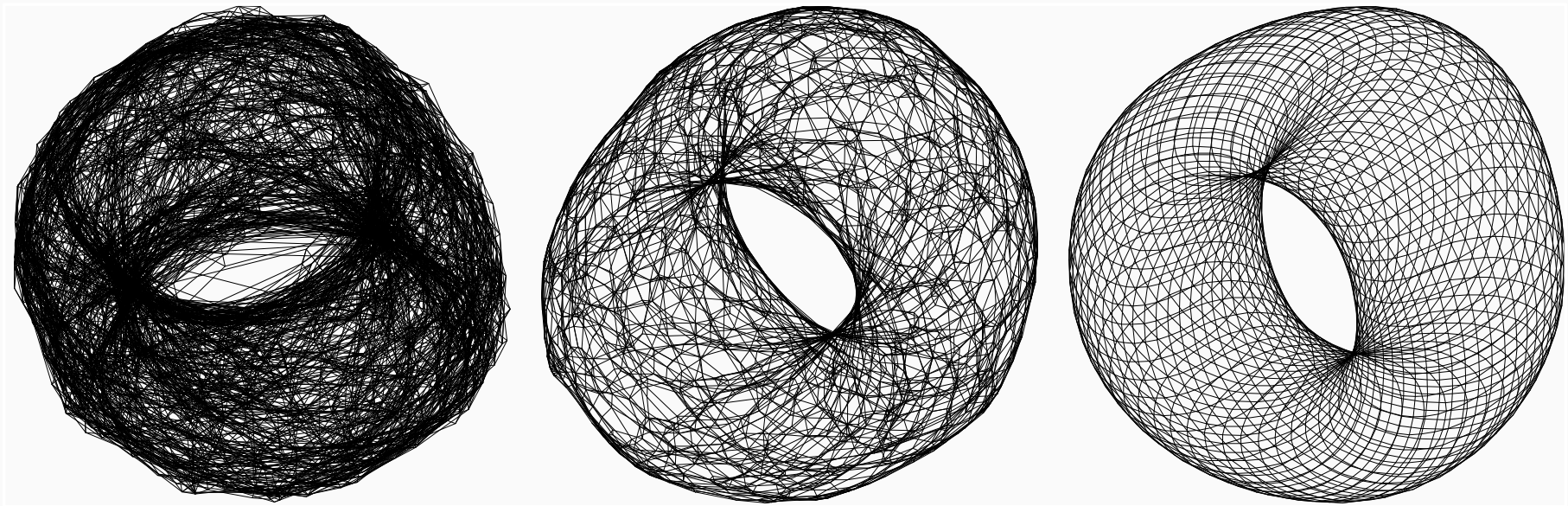
As camadas inferiores alimentam as camadas superiores com nós escolhidos aleatoriamente; a camada superior é seletiva para manter as referências.

Construindo um toro

Considere uma grade $N \times N$. Mantenha referências apenas aos vizinhos mais próximos:

$$\| (a_1, a_2) - (b_1, b_2) \| = d_1 + d_2$$

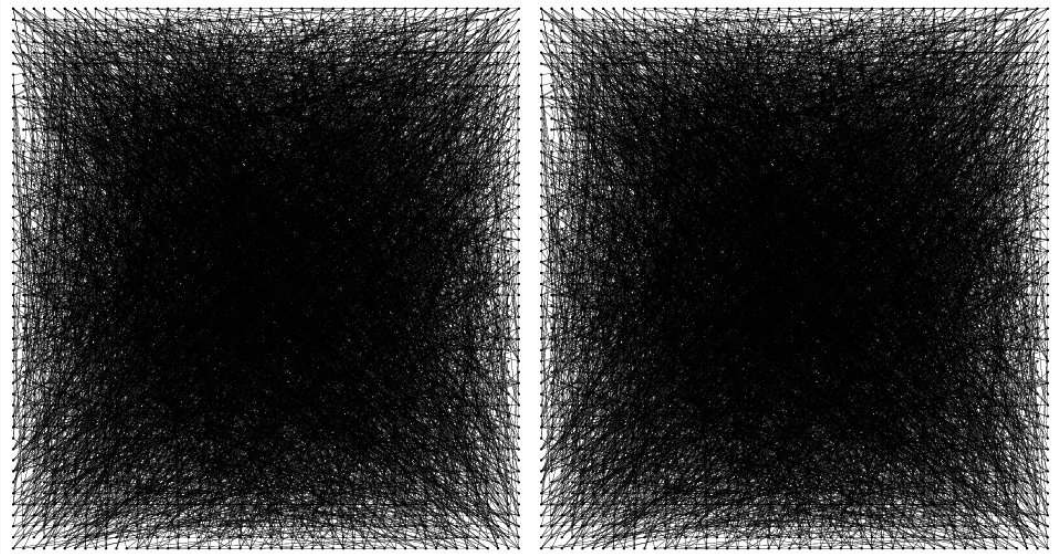
$$d_i = \mathbf{min}\{N - |a_i - b_i|, |a_i - b_i|\}$$



EXEMPLO: CRIANDO CLUSTERS DE NÓS

Ideia básica: a todo nó i é definido um *identificador de grupo* $GID(i) \in \mathbb{N}$. O objetivo é particionar o overlay em componentes disjuntos (clusters) tais que:

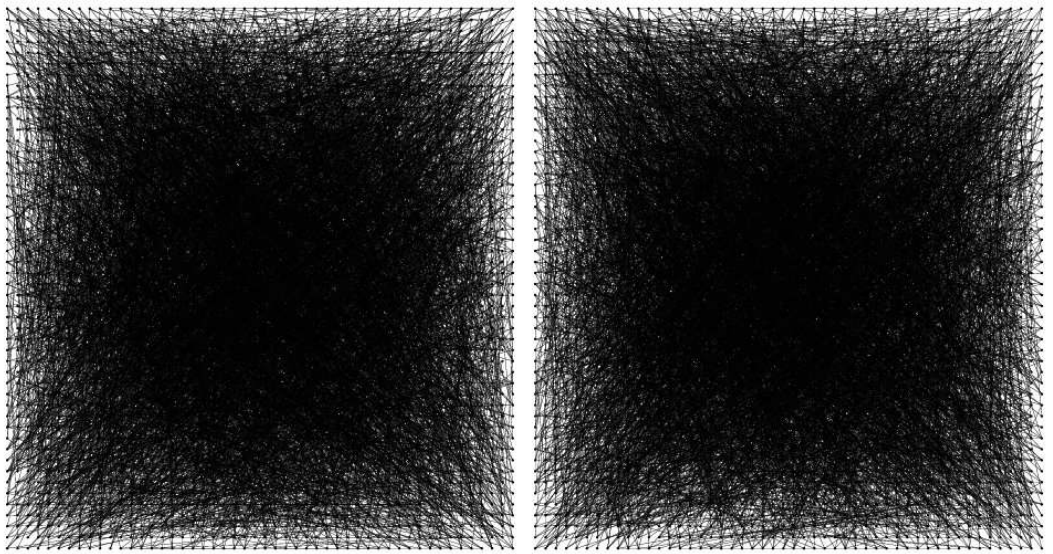
$$dist(i,j) = \begin{cases} 1 & \text{se } i \text{ e } j \text{ pertencem ao mesmo grupo } [GID(i) = GID(j)] \\ 0 & \text{caso contrário} \end{cases}$$



EXEMPLO: CRIANDO CLUSTERS DE NÓS

Ideia básica: a todo nó i é definido um *identificador de grupo* $GID(i) \in \mathbb{N}$. O objetivo é particionar o overlay em componentes disjuntos (clusters) tais que:

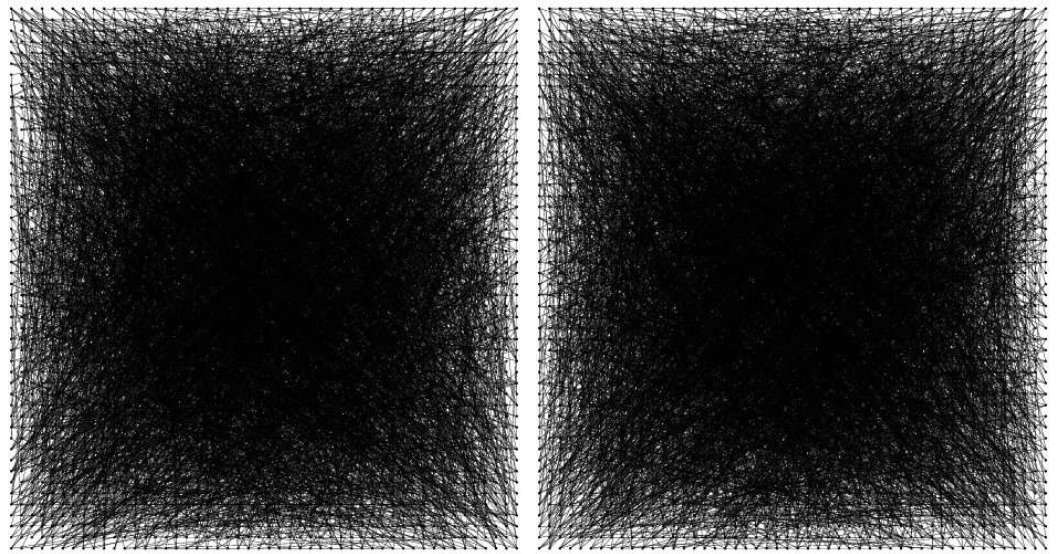
$$dist(i,j) = \begin{cases} 1 & \text{se } i \text{ e } j \text{ pertencem ao mesmo grupo } [GID(i) = GID(j)] \\ 0 & \text{caso contrário} \end{cases}$$



EXEMPLO: CRIANDO CLUSTERS DE NÓS

Ideia básica: a todo nó i é definido um *identificador de grupo* $GID(i) \in \mathbb{N}$. O objetivo é particionar o overlay em componentes disjuntos (clusters) tais que:

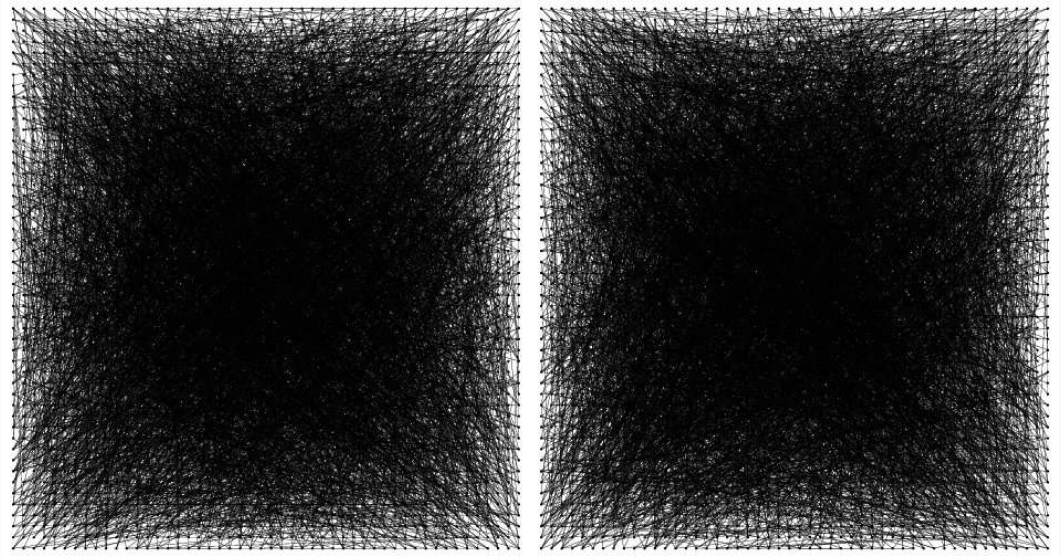
$$dist(i,j) = \begin{cases} 1 & \text{se } i \text{ e } j \text{ pertencem ao mesmo grupo } [GID(i) = GID(j)] \\ 0 & \text{caso contrário} \end{cases}$$



EXEMPLO: CRIANDO CLUSTERS DE NÓS

Ideia básica: a todo nó i é definido um *identificador de grupo* $GID(i) \in \mathbb{N}$. O objetivo é particionar o overlay em componentes disjuntos (clusters) tais que:

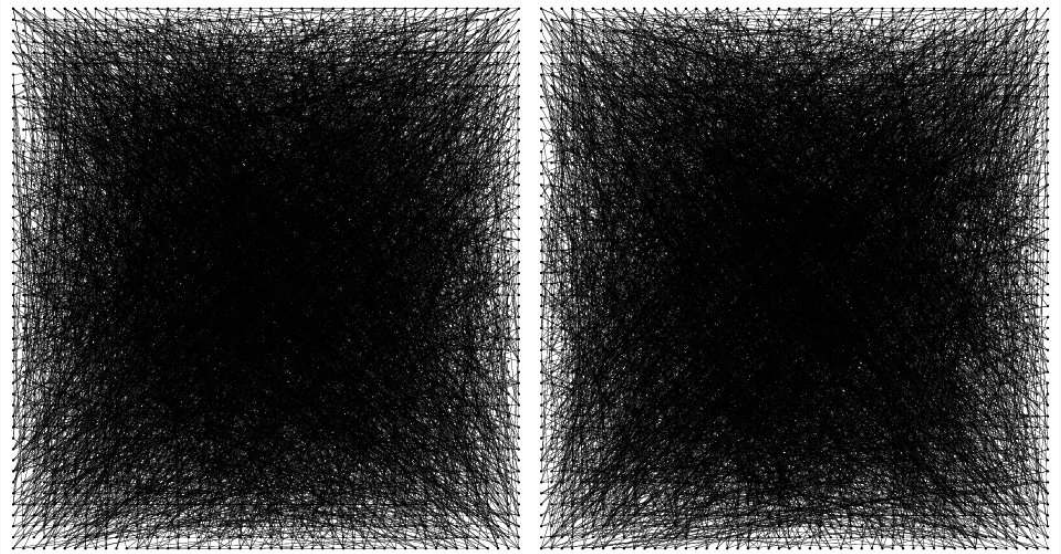
$$dist(i,j) = \begin{cases} 1 & \text{se } i \text{ e } j \text{ pertencem ao mesmo grupo } [GID(i) = GID(j)] \\ 0 & \text{caso contrário} \end{cases}$$



EXEMPLO: CRIANDO CLUSTERS DE NÓS

Ideia básica: a todo nó i é definido um *identificador de grupo* $GID(i) \in \mathbb{N}$. O objetivo é particionar o overlay em componentes disjuntos (clusters) tais que:

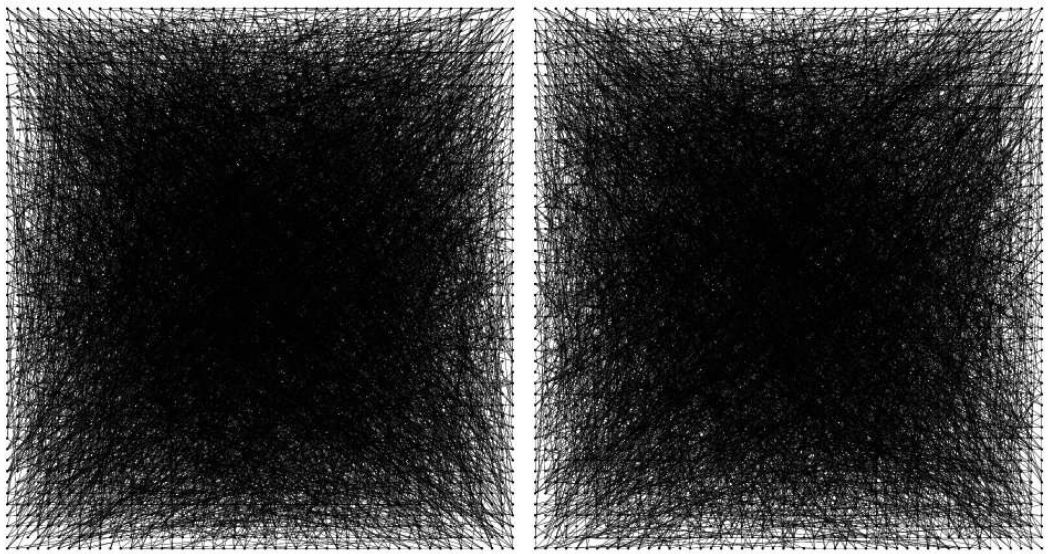
$$dist(i,j) = \begin{cases} 1 & \text{se } i \text{ e } j \text{ pertencem ao mesmo grupo } [GID(i) = GID(j)] \\ 0 & \text{caso contrário} \end{cases}$$



EXEMPLO: CRIANDO CLUSTERS DE NÓS

Ideia básica: a todo nó i é definido um *identificador de grupo* $GID(i) \in \mathbb{N}$. O objetivo é particionar o overlay em componentes disjuntos (clusters) tais que:

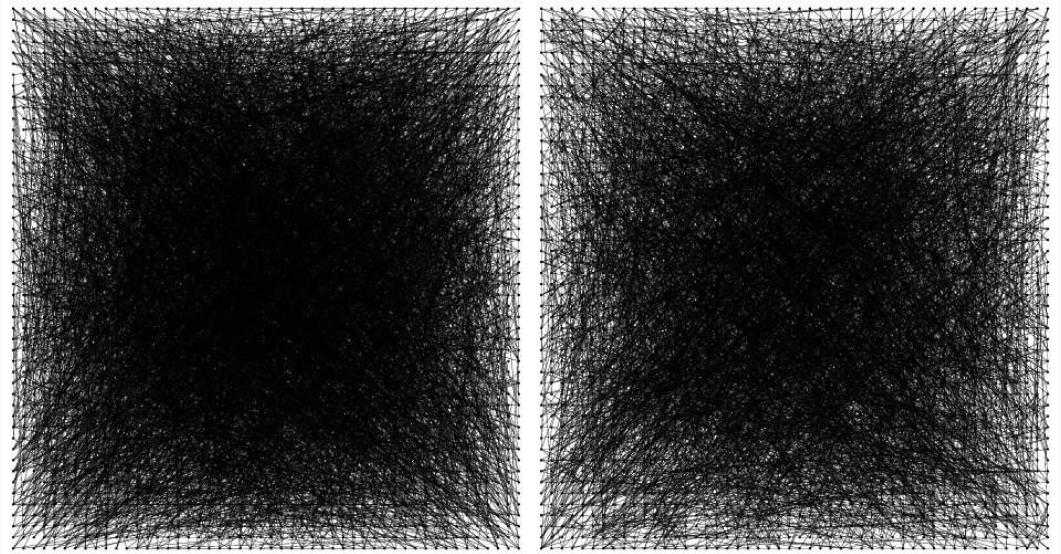
$$dist(i,j) = \begin{cases} 1 & \text{se } i \text{ e } j \text{ pertencem ao mesmo grupo } [GID(i) = GID(j)] \\ 0 & \text{caso contrário} \end{cases}$$



EXEMPLO: CRIANDO CLUSTERS DE NÓS

Ideia básica: a todo nó i é definido um *identificador de grupo* $GID(i) \in \mathbb{N}$. O objetivo é particionar o overlay em componentes disjuntos (clusters) tais que:

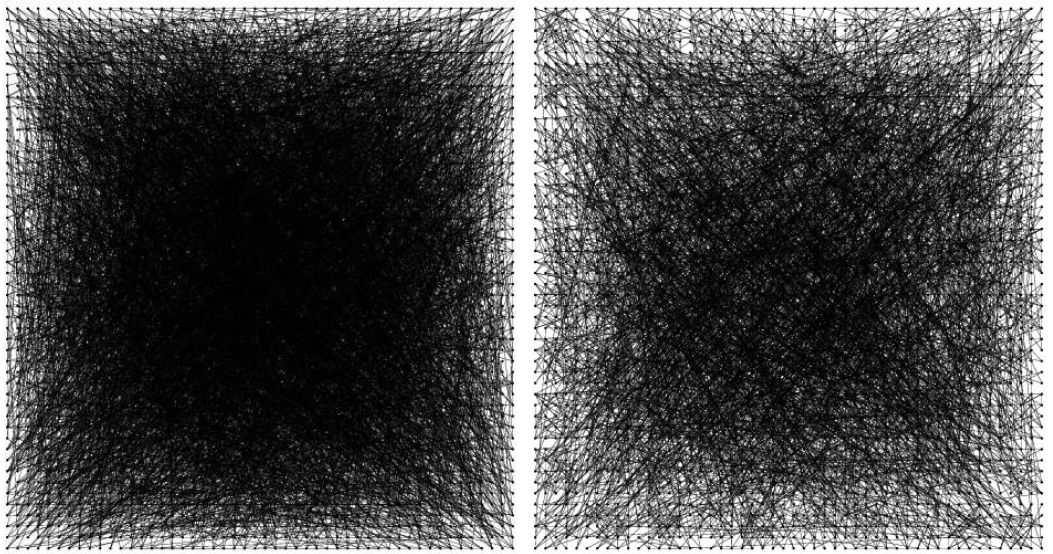
$$dist(i,j) = \begin{cases} 1 & \text{se } i \text{ e } j \text{ pertencem ao mesmo grupo } [GID(i) = GID(j)] \\ 0 & \text{caso contrário} \end{cases}$$



EXEMPLO: CRIANDO CLUSTERS DE NÓS

Ideia básica: a todo nó i é definido um *identificador de grupo* $GID(i) \in \mathbb{N}$. O objetivo é particionar o overlay em componentes disjuntos (clusters) tais que:

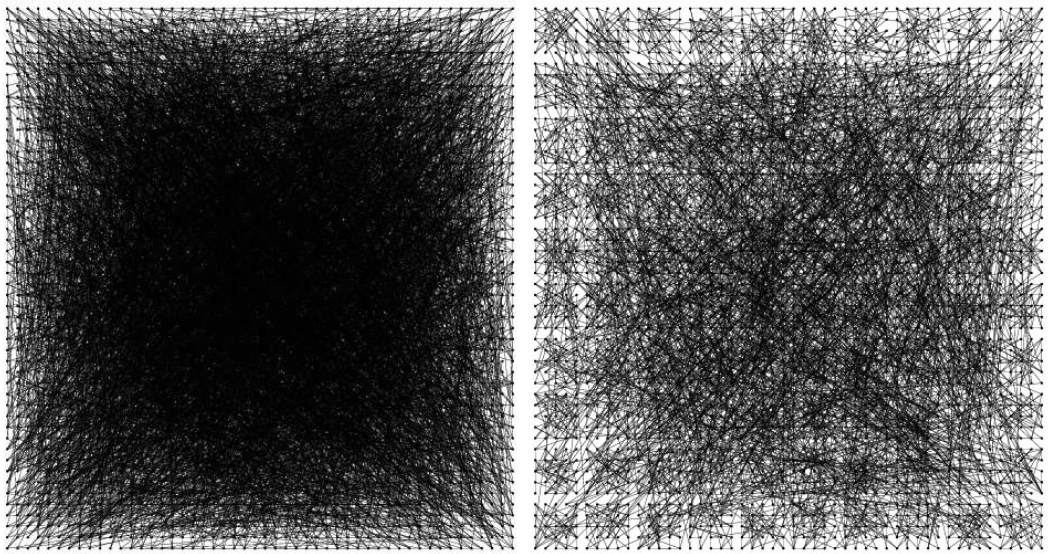
$$dist(i,j) = \begin{cases} 1 & \text{se } i \text{ e } j \text{ pertencem ao mesmo grupo } [GID(i) = GID(j)] \\ 0 & \text{caso contrário} \end{cases}$$



EXEMPLO: CRIANDO CLUSTERS DE NÓS

Ideia básica: a todo nó i é definido um *identificador de grupo* $GID(i) \in \mathbb{N}$. O objetivo é particionar o overlay em componentes disjuntos (clusters) tais que:

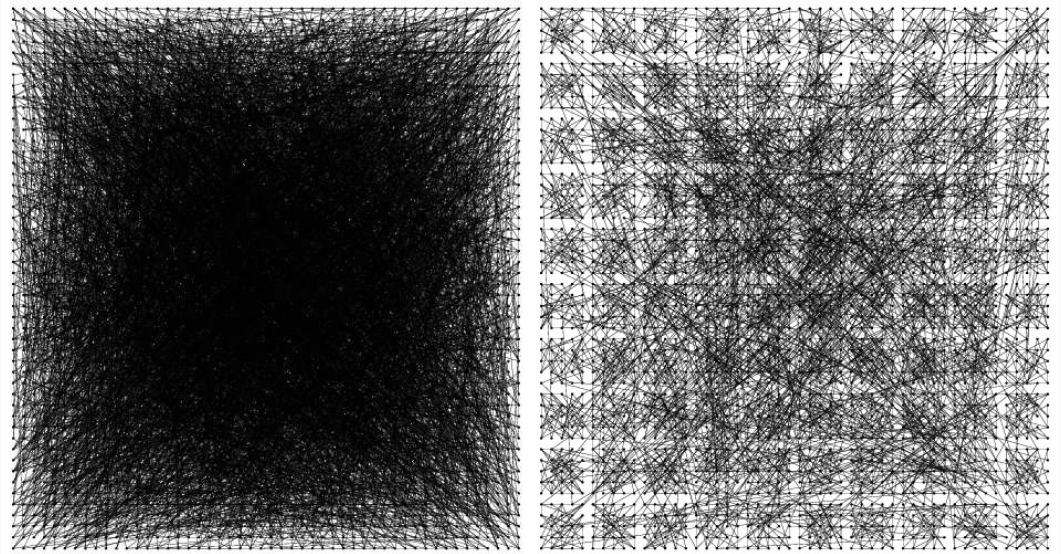
$$dist(i,j) = \begin{cases} 1 & \text{se } i \text{ e } j \text{ pertencem ao mesmo grupo } [GID(i) = GID(j)] \\ 0 & \text{caso contrário} \end{cases}$$



EXEMPLO: CRIANDO CLUSTERS DE NÓS

Ideia básica: a todo nó i é definido um *identificador de grupo* $GID(i) \in \mathbb{N}$. O objetivo é particionar o overlay em componentes disjuntos (clusters) tais que:

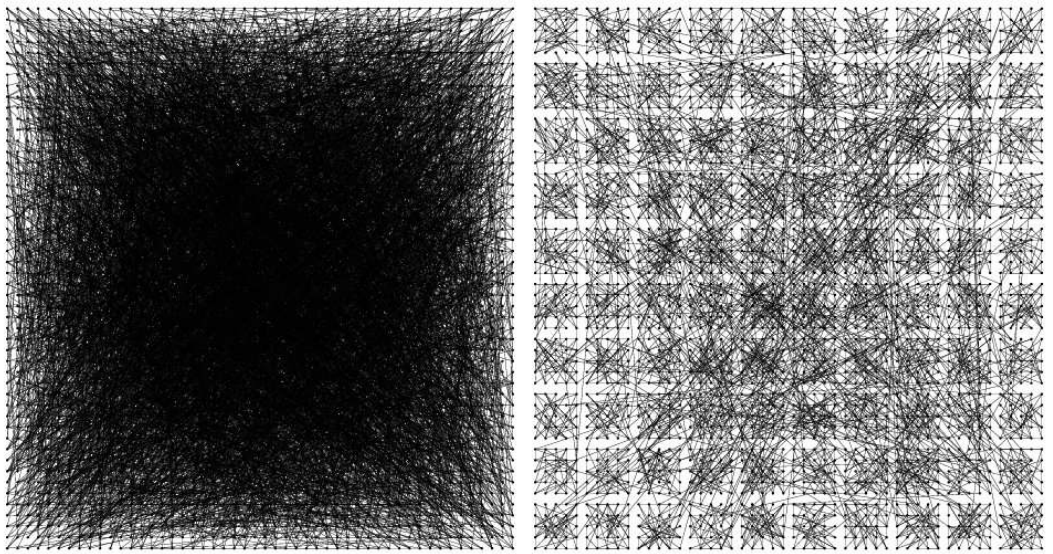
$$dist(i,j) = \begin{cases} 1 & \text{se } i \text{ e } j \text{ pertencem ao mesmo grupo } [GID(i) = GID(j)] \\ 0 & \text{caso contrário} \end{cases}$$



EXEMPLO: CRIANDO CLUSTERS DE NÓS

Ideia básica: a todo nó i é definido um *identificador de grupo* $GID(i) \in \mathbb{N}$. O objetivo é particionar o overlay em componentes disjuntos (clusters) tais que:

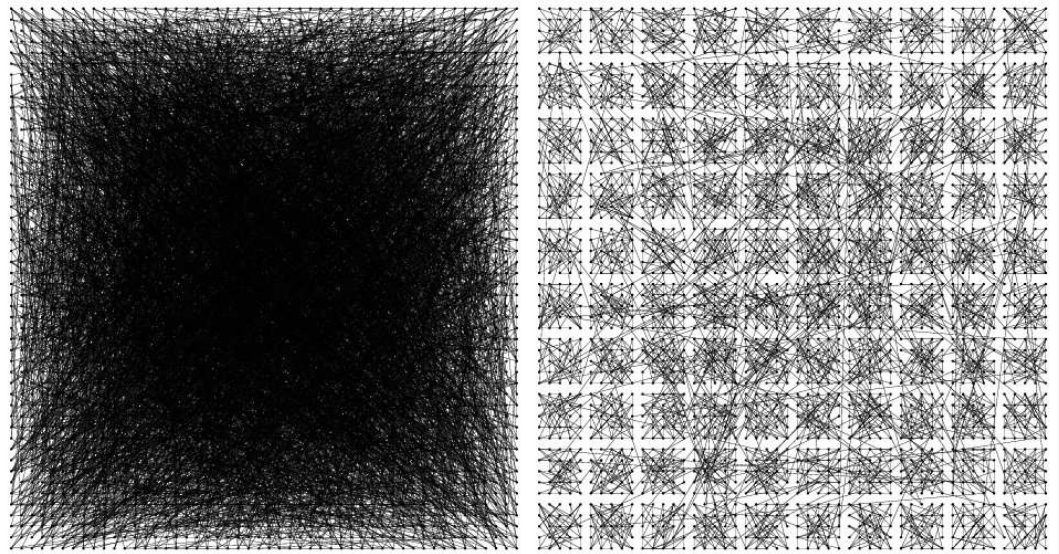
$$dist(i,j) = \begin{cases} 1 & \text{se } i \text{ e } j \text{ pertencem ao mesmo grupo } [GID(i) = GID(j)] \\ 0 & \text{caso contrário} \end{cases}$$



EXEMPLO: CRIANDO CLUSTERS DE NÓS

Ideia básica: a todo nó i é definido um *identificador de grupo* $GID(i) \in \mathbb{N}$. O objetivo é particionar o overlay em componentes disjuntos (clusters) tais que:

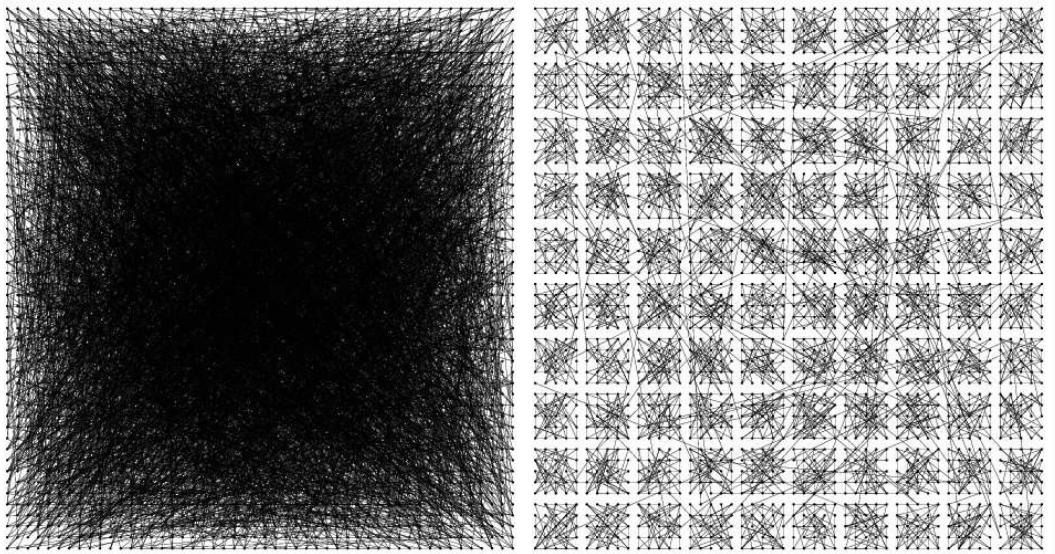
$$dist(i,j) = \begin{cases} 1 & \text{se } i \text{ e } j \text{ pertencem ao mesmo grupo } [GID(i) = GID(j)] \\ 0 & \text{caso contrário} \end{cases}$$



EXEMPLO: CRIANDO CLUSTERS DE NÓS

Ideia básica: a todo nó i é definido um *identificador de grupo* $GID(i) \in \mathbb{N}$. O objetivo é particionar o overlay em componentes disjuntos (clusters) tais que:

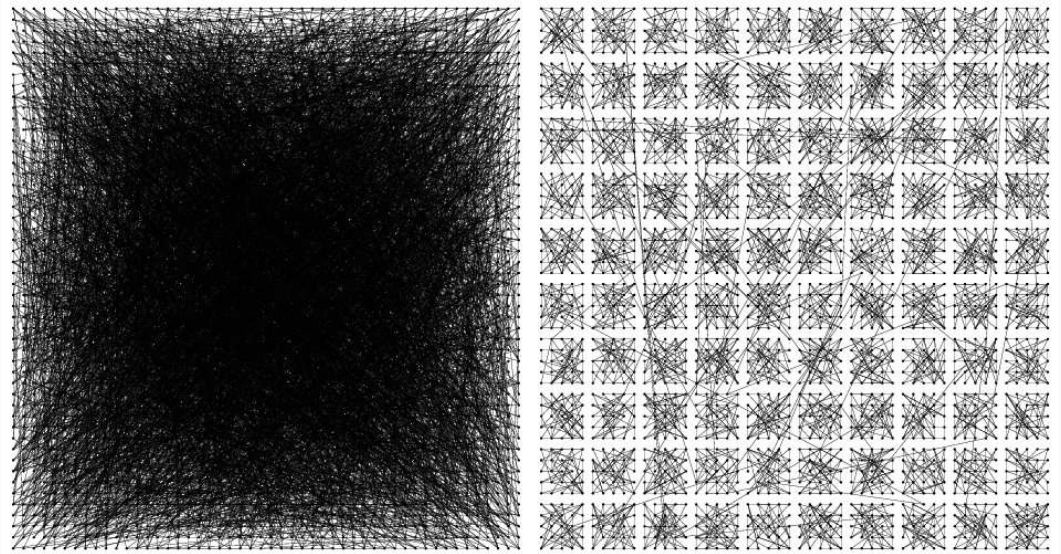
$$dist(i,j) = \begin{cases} 1 & \text{se } i \text{ e } j \text{ pertencem ao mesmo grupo } [GID(i) = GID(j)] \\ 0 & \text{caso contrário} \end{cases}$$



EXEMPLO: CRIANDO CLUSTERS DE NÓS

Ideia básica: a todo nó i é definido um *identificador de grupo* $GID(i) \in \mathbb{N}$. O objetivo é particionar o overlay em componentes disjuntos (clusters) tais que:

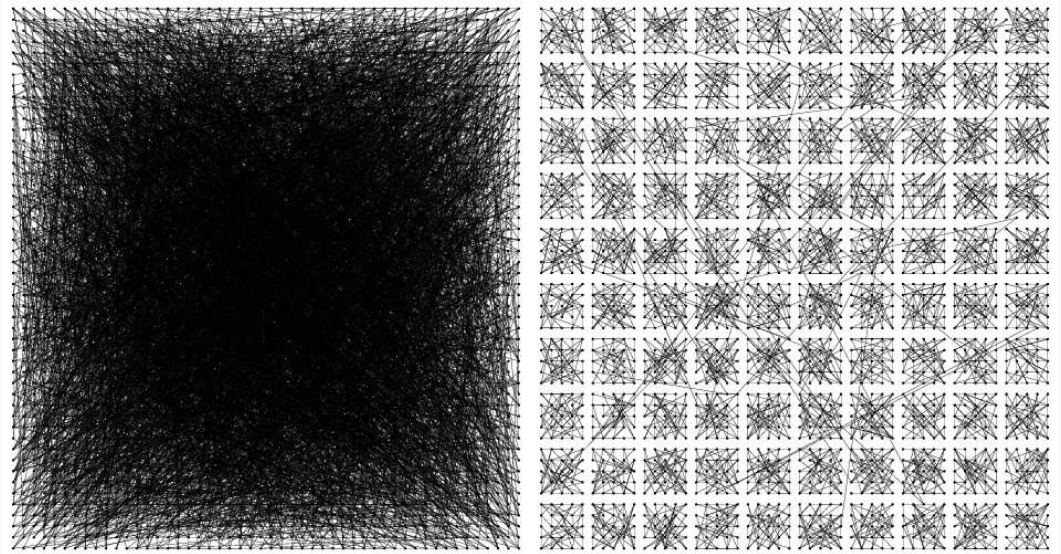
$$dist(i,j) = \begin{cases} 1 & \text{se } i \text{ e } j \text{ pertencem ao mesmo grupo } [GID(i) = GID(j)] \\ 0 & \text{caso contrário} \end{cases}$$



EXEMPLO: CRIANDO CLUSTERS DE NÓS

Ideia básica: a todo nó i é definido um *identificador de grupo* $GID(i) \in \mathbb{N}$. O objetivo é particionar o overlay em componentes disjuntos (clusters) tais que:

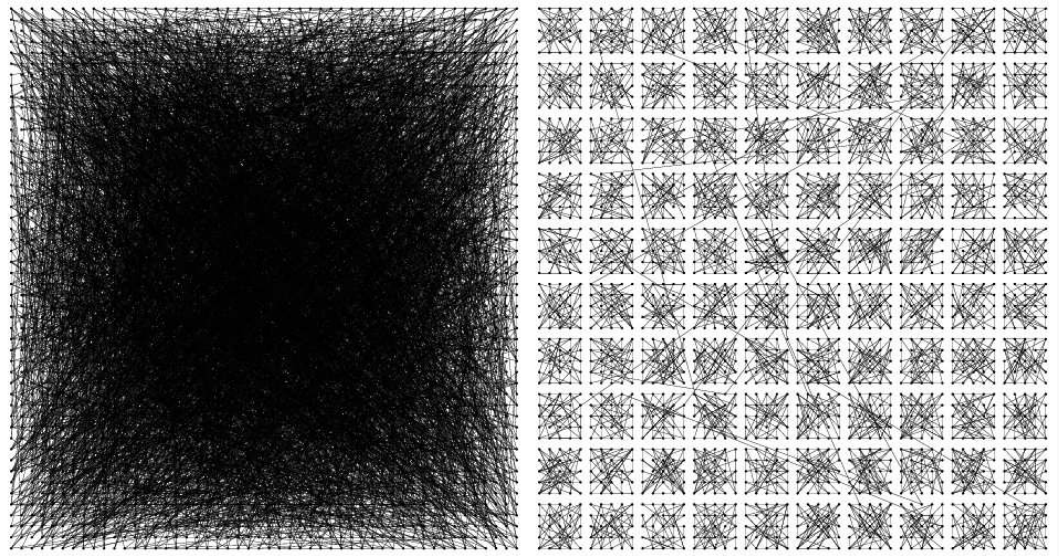
$$dist(i,j) = \begin{cases} 1 & \text{se } i \text{ e } j \text{ pertencem ao mesmo grupo } [GID(i) = GID(j)] \\ 0 & \text{caso contrário} \end{cases}$$



EXEMPLO: CRIANDO CLUSTERS DE NÓS

Ideia básica: a todo nó i é definido um *identificador de grupo* $GID(i) \in \mathbb{N}$. O objetivo é particionar o overlay em componentes disjuntos (clusters) tais que:

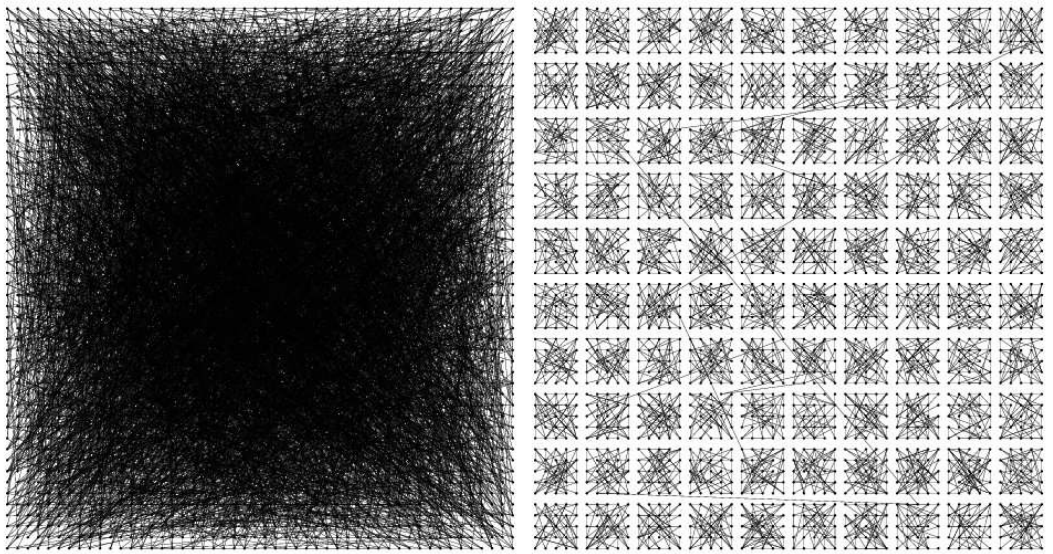
$$dist(i,j) = \begin{cases} 1 & \text{se } i \text{ e } j \text{ pertencem ao mesmo grupo } [GID(i) = GID(j)] \\ 0 & \text{caso contrário} \end{cases}$$



EXEMPLO: CRIANDO CLUSTERS DE NÓS

Ideia básica: a todo nó i é definido um *identificador de grupo* $GID(i) \in \mathbb{N}$. O objetivo é particionar o overlay em componentes disjuntos (clusters) tais que:

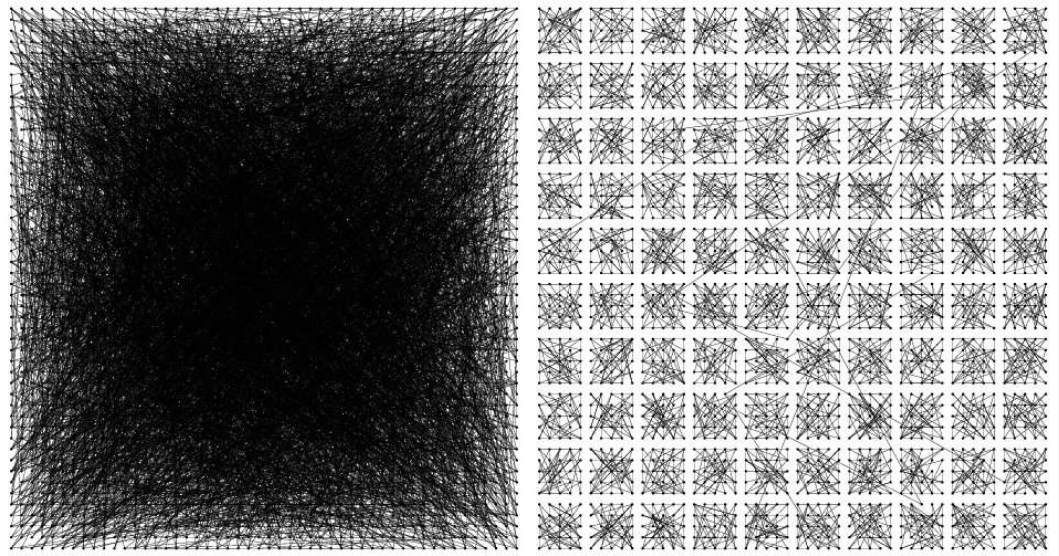
$$dist(i,j) = \begin{cases} 1 & \text{se } i \text{ e } j \text{ pertencem ao mesmo grupo } [GID(i) = GID(j)] \\ 0 & \text{caso contrário} \end{cases}$$



EXEMPLO: CRIANDO CLUSTERS DE NÓS

Ideia básica: a todo nó i é definido um *identificador de grupo* $GID(i) \in \mathbb{N}$. O objetivo é particionar o overlay em componentes disjuntos (clusters) tais que:

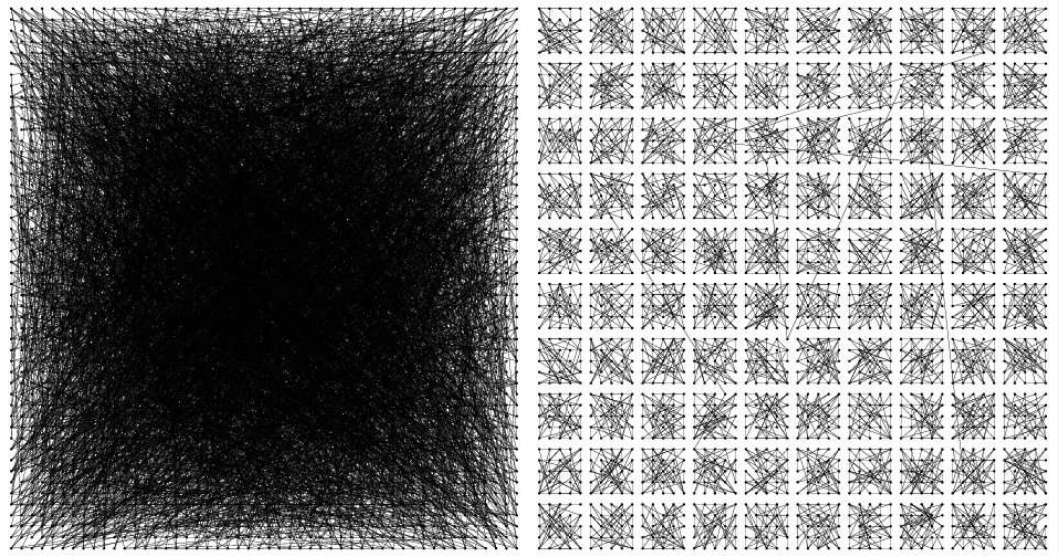
$$dist(i,j) = \begin{cases} 1 & \text{se } i \text{ e } j \text{ pertencem ao mesmo grupo } [GID(i) = GID(j)] \\ 0 & \text{caso contrário} \end{cases}$$



EXEMPLO: CRIANDO CLUSTERS DE NÓS

Ideia básica: a todo nó i é definido um *identificador de grupo* $GID(i) \in \mathbb{N}$. O objetivo é particionar o overlay em componentes disjuntos (clusters) tais que:

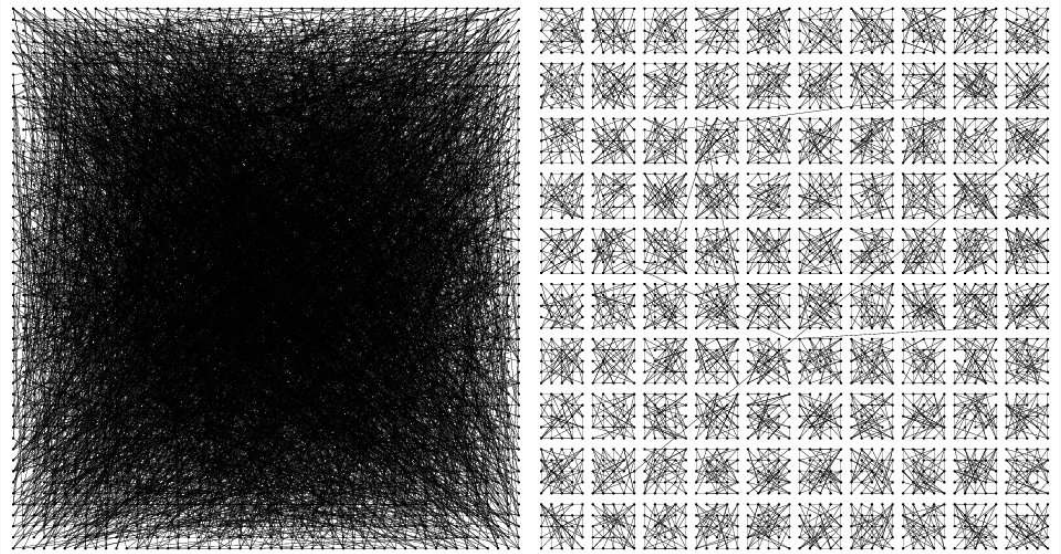
$$dist(i,j) = \begin{cases} 1 & \text{se } i \text{ e } j \text{ pertencem ao mesmo grupo } [GID(i) = GID(j)] \\ 0 & \text{caso contrário} \end{cases}$$



EXEMPLO: CRIANDO CLUSTERS DE NÓS

Ideia básica: a todo nó i é definido um *identificador de grupo* $GID(i) \in \mathbb{N}$. O objetivo é particionar o overlay em componentes disjuntos (clusters) tais que:

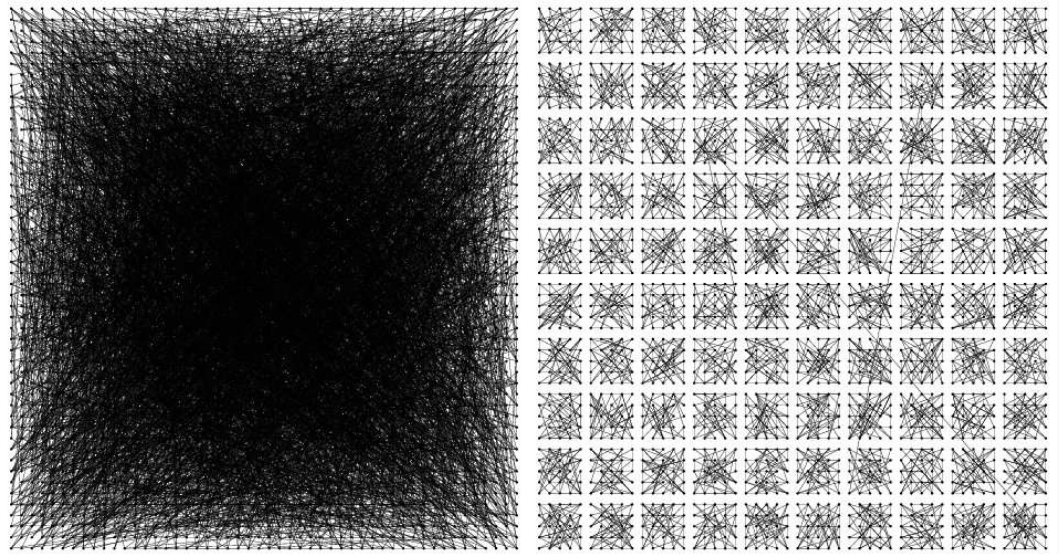
$$dist(i,j) = \begin{cases} 1 & \text{se } i \text{ e } j \text{ pertencem ao mesmo grupo } [GID(i) = GID(j)] \\ 0 & \text{caso contrário} \end{cases}$$



EXEMPLO: CRIANDO CLUSTERS DE NÓS

Ideia básica: a todo nó i é definido um *identificador de grupo* $GID(i) \in \mathbb{N}$. O objetivo é particionar o overlay em componentes disjuntos (clusters) tais que:

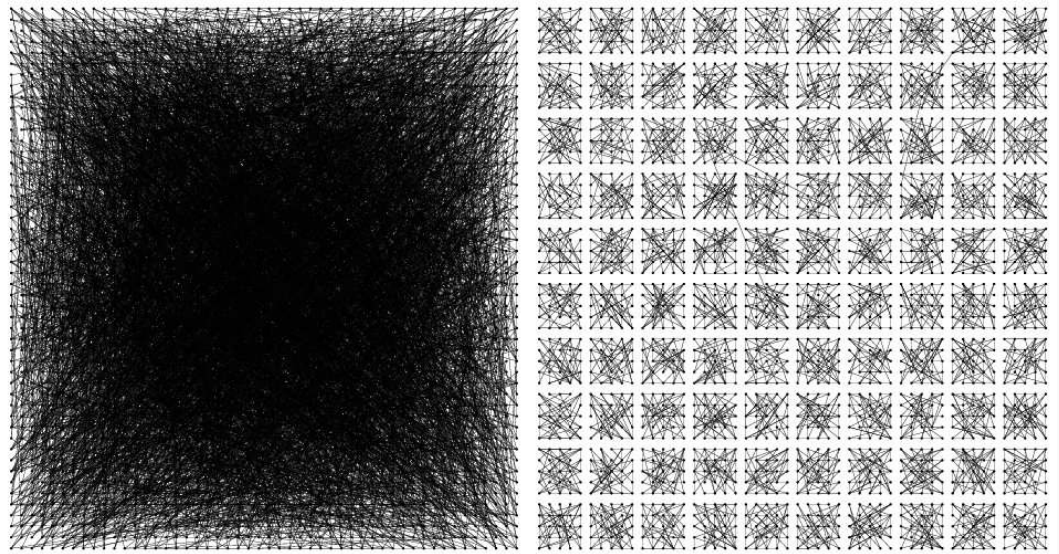
$$dist(i,j) = \begin{cases} 1 & \text{se } i \text{ e } j \text{ pertencem ao mesmo grupo } [GID(i) = GID(j)] \\ 0 & \text{caso contrário} \end{cases}$$



EXEMPLO: CRIANDO CLUSTERS DE NÓS

Ideia básica: a todo nó i é definido um *identificador de grupo* $GID(i) \in \mathbb{N}$. O objetivo é particionar o overlay em componentes disjuntos (clusters) tais que:

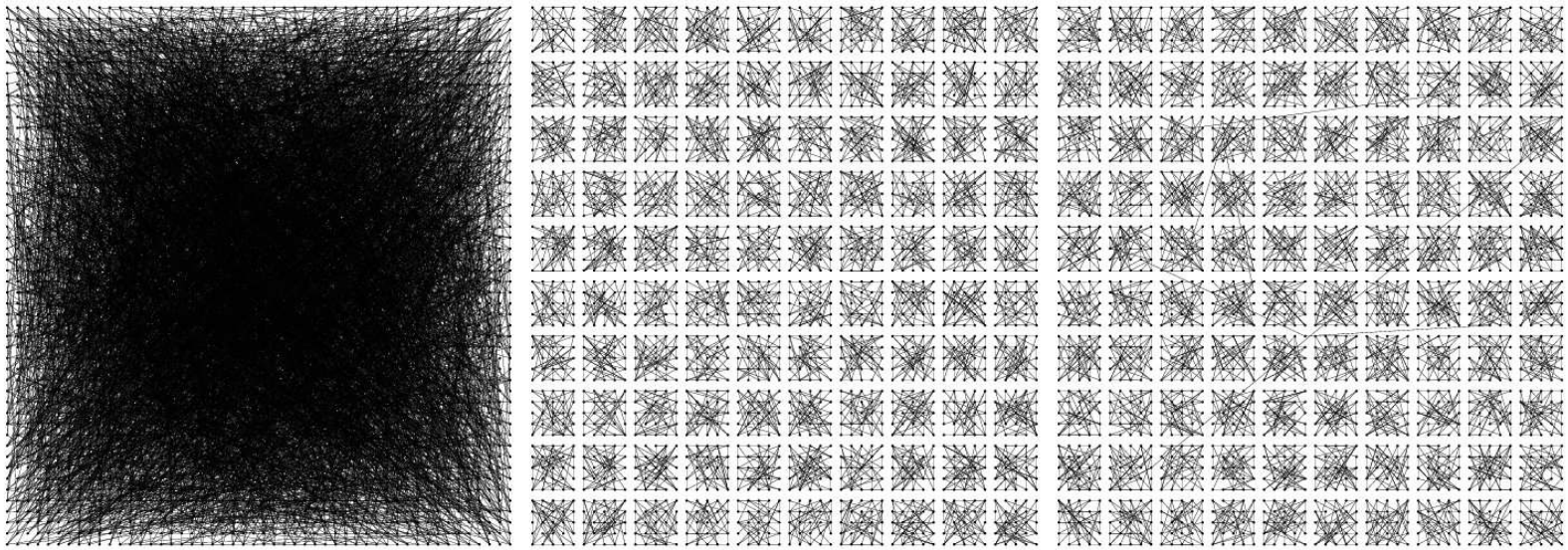
$$dist(i,j) = \begin{cases} 1 & \text{se } i \text{ e } j \text{ pertencem ao mesmo grupo } [GID(i) = GID(j)] \\ 0 & \text{caso contrário} \end{cases}$$



EXEMPLO: CRIANDO CLUSTERS DE NÓS

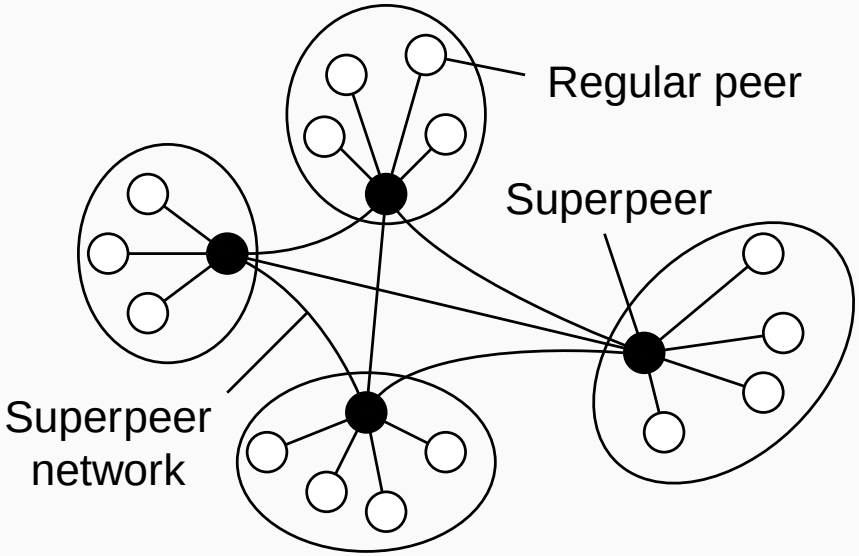
Ideia básica: a todo nó i é definido um *identificador de grupo* $GID(i) \in \mathbb{N}$. O objetivo é particionar o overlay em componentes disjuntos (clusters) tais que:

$$dist(i,j) = \begin{cases} 1 & \text{se } i \text{ e } j \text{ pertencem ao mesmo grupo } [GID(i) = GID(j)] \\ 0 & \text{caso contrário} \end{cases}$$



Observação

Às vezes, selecionar alguns nós para realizar algum trabalho específico pode ser útil.



Exemplos:

- Peers para manter um índice (para buscas)
- Peers para monitorar o estado da rede
- Peers capazes de configurar conexões

Princípio de operação do Skype: A quer contactar B

Tanto A quanto B estão na Internet pública

- Uma conexão TCP é estabelecida entre A e B para envio de pacotes de controle
- A chamada real usa pacotes UDP entre as portas negociadas

Princípio de operação do Skype: A quer contactar B

Tanto A quanto B estão na Internet pública

- Uma conexão TCP é estabelecida entre A e B para envio de pacotes de controle
- A chamada real usa pacotes UDP entre as portas negociadas

A está atrás de um firewall, B está na Internet pública

- A configura uma conexão TCP (para os pacotes de controle) com um superpeer S
- S configura uma conexão TCP (para redirecionar os pacotes de controle) com B
- A chamada real usa pacotes UDP diretamente entre A e B

Princípio de operação do Skype: A quer contactar B

Tanto A quanto B estão na Internet pública

- Uma conexão TCP é estabelecida entre A e B para envio de pacotes de controle
- A chamada real usa pacotes UDP entre as portas negociadas

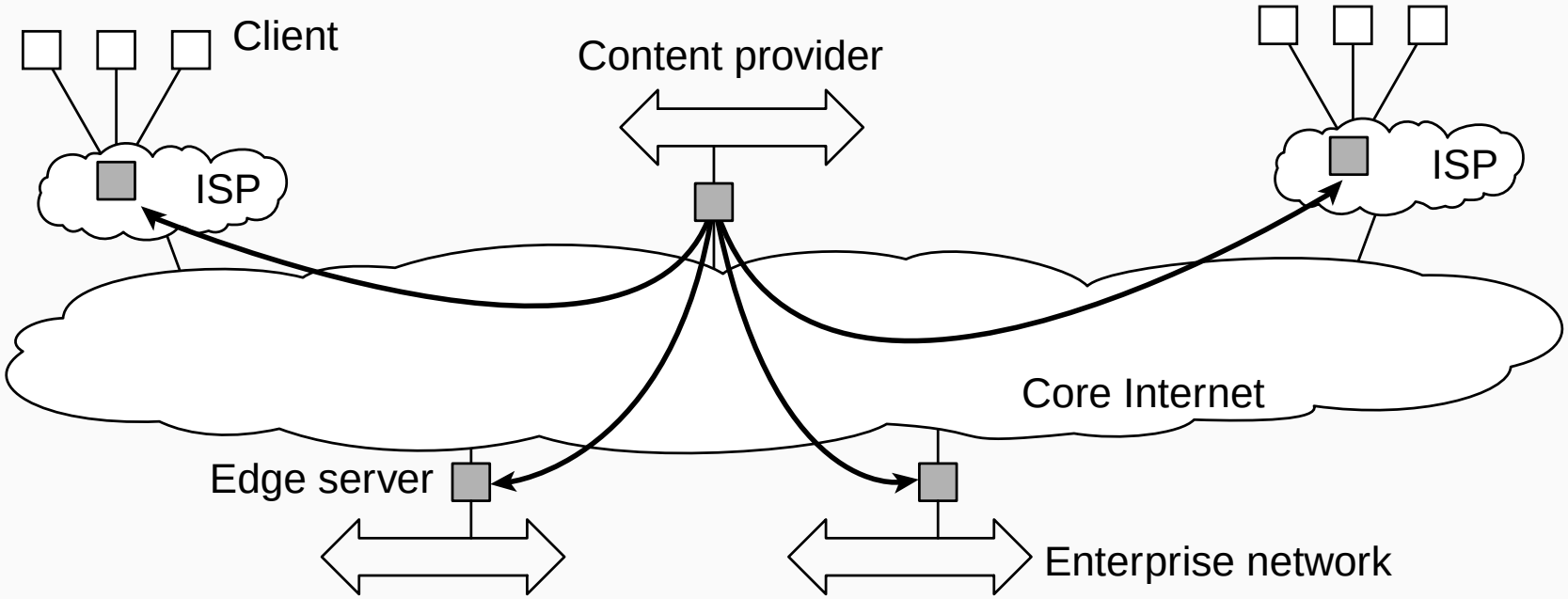
A está atrás de um firewall, B está na Internet pública

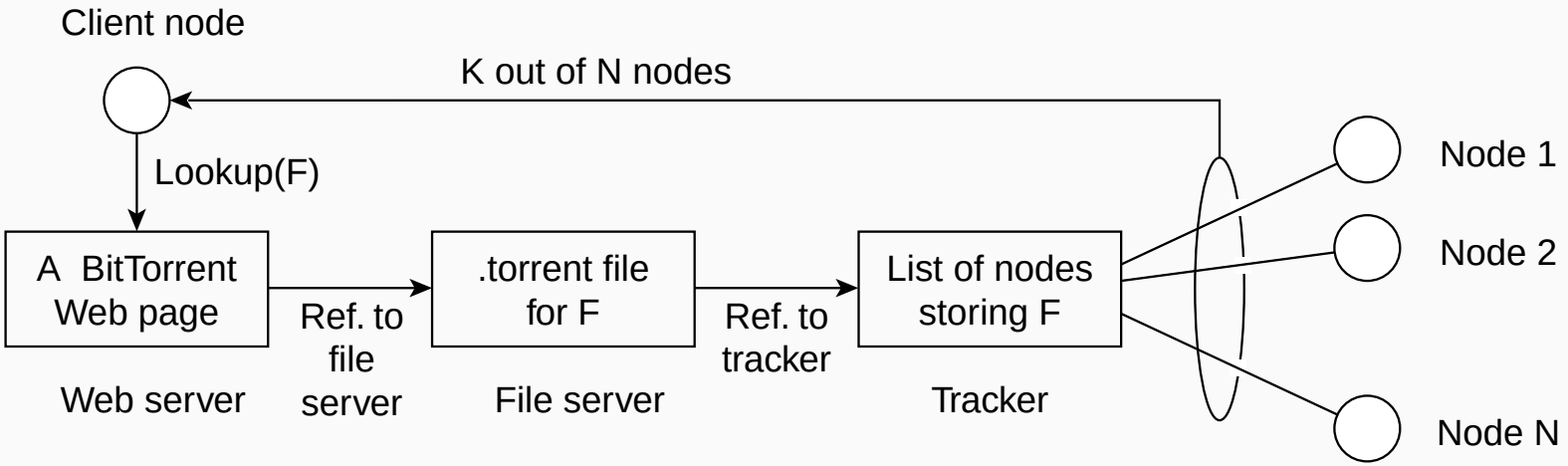
- A configura uma conexão TCP (para os pacotes de controle) com um superpeer S
- S configura uma conexão TCP (para redirecionar os pacotes de controle) com B
- A chamada real usa pacotes UDP diretamente entre A e B

Tanto A quanto B estão atrás de um firewall

- A conecta com um superpeer S via TCP
- S configura uma conexão TCP com B
- Para a chamada real, outro superpeer é usado para funcionar como retransmissor (relay): A (e B) configura a conexão com R
- A chamada é encaminhada usando duas conexões TCP, usando R como intermediário

Exemplo:
Arquiteturas de servidores de borda (*edge-server*), utilizados com frequência como Content Delivery Networks (redes de distribuição de conteúdo).





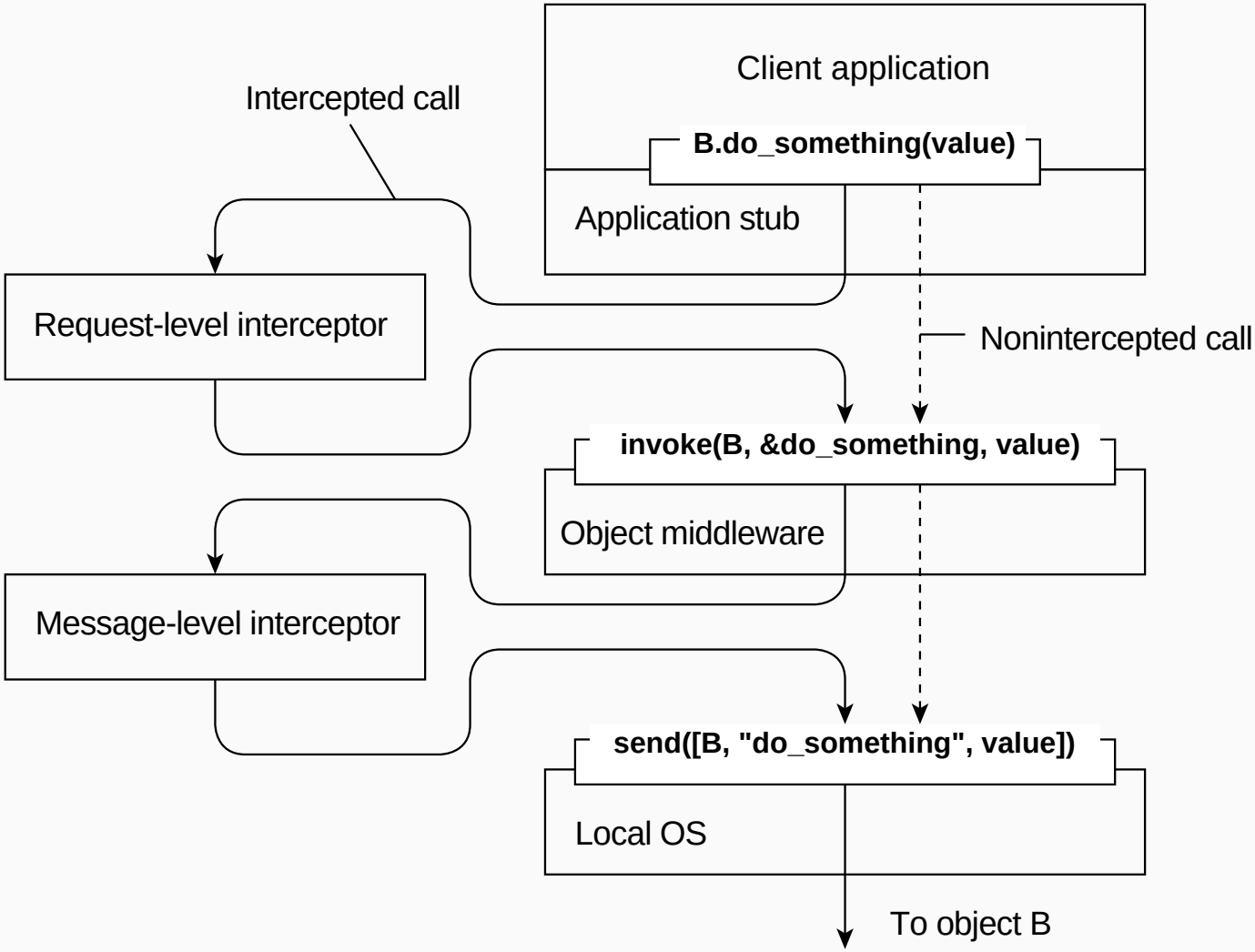
Ideia básica

Assim que um nó identifica de onde o arquivo será baixado, ele se junta a uma swarm (multidão) de pessoas que, em paralelo, receberão pedaços do arquivo da fonte e redistribuirão esses pedaços entre os outros.

Problema
Em muitos casos, arquiteturas/sistemas distribuídos são desenvolvidos de acordo com um estilo arquitetural específico. O estilo escolhido pode não ser o melhor em todos os casos $\Rightarrow$ é necessário adaptar o comportamento do middleware (dinamicamente).

Interceptors
Interceptam o fluxo de controle normal quando um objeto remoto for invocado.

INTERCEPTORS



- Separação de interesses: tente separar as funcionalidades extras e depois costurá-las em uma única implementação ⇒ aplicabilidade restrita (*toy examples*)
- Reflexão computacional: deixe o programa inspecionar-se em tempo de execução e adaptar/mudar suas configurações dinamicamente, se necessário ⇒ ocorre principalmente no nível da linguagem, aplicabilidade não é muito clara.
- Projeto baseado em componentes: organize uma aplicação distribuída em componentes que podem ser substituídos dinamicamente quando necessário ⇒ causa muitas e complexas interdependências entre componentes.

Observação
A distinção entre arquiteturas de sistemas e arquiteturas de software fica confusa quando adaptação automática deve ser considerada:

- Autoconfiguração
- Autogerenciamento
- Autocura
- Auto-otimização
- Auto-*

MODELO DE REGULAÇÃO POR FEEDBACK

Em muitos casos, sistemas auto-* são organizados como um sistema de regulação por feedback