

# Análise Forense de Documentos Digitais

*Prof. Dr. Anderson Rocha*

[anderson.rocha@ic.unicamp.br](mailto:anderson.rocha@ic.unicamp.br)

<http://www.ic.unicamp.br/~rocha>

---

Reasoning for Complex Data (RECOD) Lab.  
Institute of Computing, Unicamp

Av. Albert Einstein, 1251 – Cidade Universitária  
CEP 13083-970 • Campinas/SP – Brasil

---

# **File Carving e Smart File Carving**

Baseado em “The evolution of file carving – the benefits and problems of forensics recovery.” **Anandabrata Pal and Nasir Memon.** IEEE Signal Processing Magazine, 26(2):59–71, March 2009.

# Organização

# Organização

- ▶ Introdução e conceitos gerais
- ▶ Recuperação de arquivos tradicional
- ▶ File Carving
- ▶ Smart Carving
- ▶ Conclusão
- ▶ Referências

# **Introdução e Conceitos Gerais**

# O que é file carving?

- ▶ É a extração e recuperação de arquivos baseada somente em sua estrutura

# Por que File Carving?

A enorme massa de dados no formato digital está sujeita à:

- ▶ Corrupção do sistema de arquivos
- ▶ Formatação de um dispositivo
- ▶ Dados em formato proprietário ou desconhecido
- ▶ Arquivos removidos/apagados (intencionalmente ou não)



# Armazenamento (1)

- ▶ Discos Rígidos e SSD's são divididos em Clusters
- ▶ Clusters são compostos por Setores e representam a menor quantidade que é possível gravar em uma única vez
- ▶ Setores costumam ter 512 ou 4K bytes

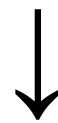
# Armazenamento (2)

- ▶ Os sistemas de arquivos:
  - Gerenciam os arquivos
  - Alocam Blocos (Clusters)
- ▶ A alocação de blocos pode ou não ser sequencial
  - Alocação não sequencial indica fragmentação

# Exemplo: Armazenamento

**Arquivo**

Dados de um arquivo sob o ponto de vista de aplicações



A1

A2

A3

A4

A5

A6

A7

A8

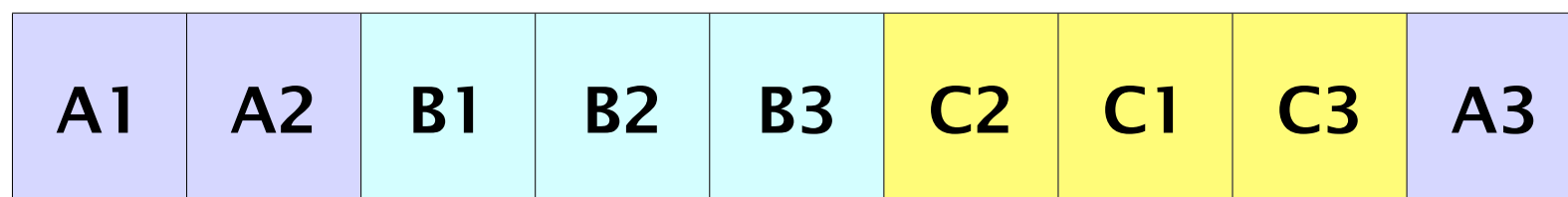
A9

Dados de um arquivo no disco, dividido em blocos

# Fragmentação

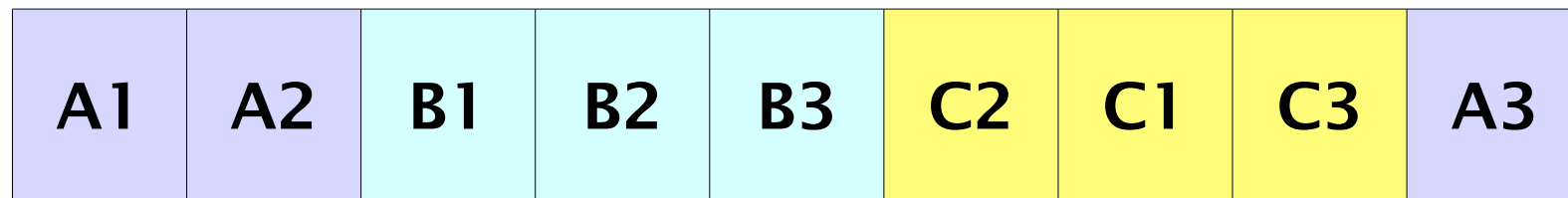
- ▶ O nível de fragmentação é dependente do:
  - Sistema de arquivos
  - Tamanho do arquivo
  - Tamanho do bloco
- ▶ Alocação não sequencial indica fragmentação
- ▶ Os fragmentos podem estar dispostos em qualquer ordem

# Exemplo de Fragmentação



No exemplo, cada célula representa um bloco. Estão representados 3 arquivos, cada um constituído por 3 fragmentos. Os fragmentos 1, 2 e 3 representam, respectivamente: o início, meio e fim do arquivo.

# Nomenclatura



- Utilizando o arquivo A como exemplo:
  - A1 e A2 são o fragmento base
  - A2 é o ponto de fragmentação

# **Recuperação de arquivos tradicional**

# Recuperação Tradicional

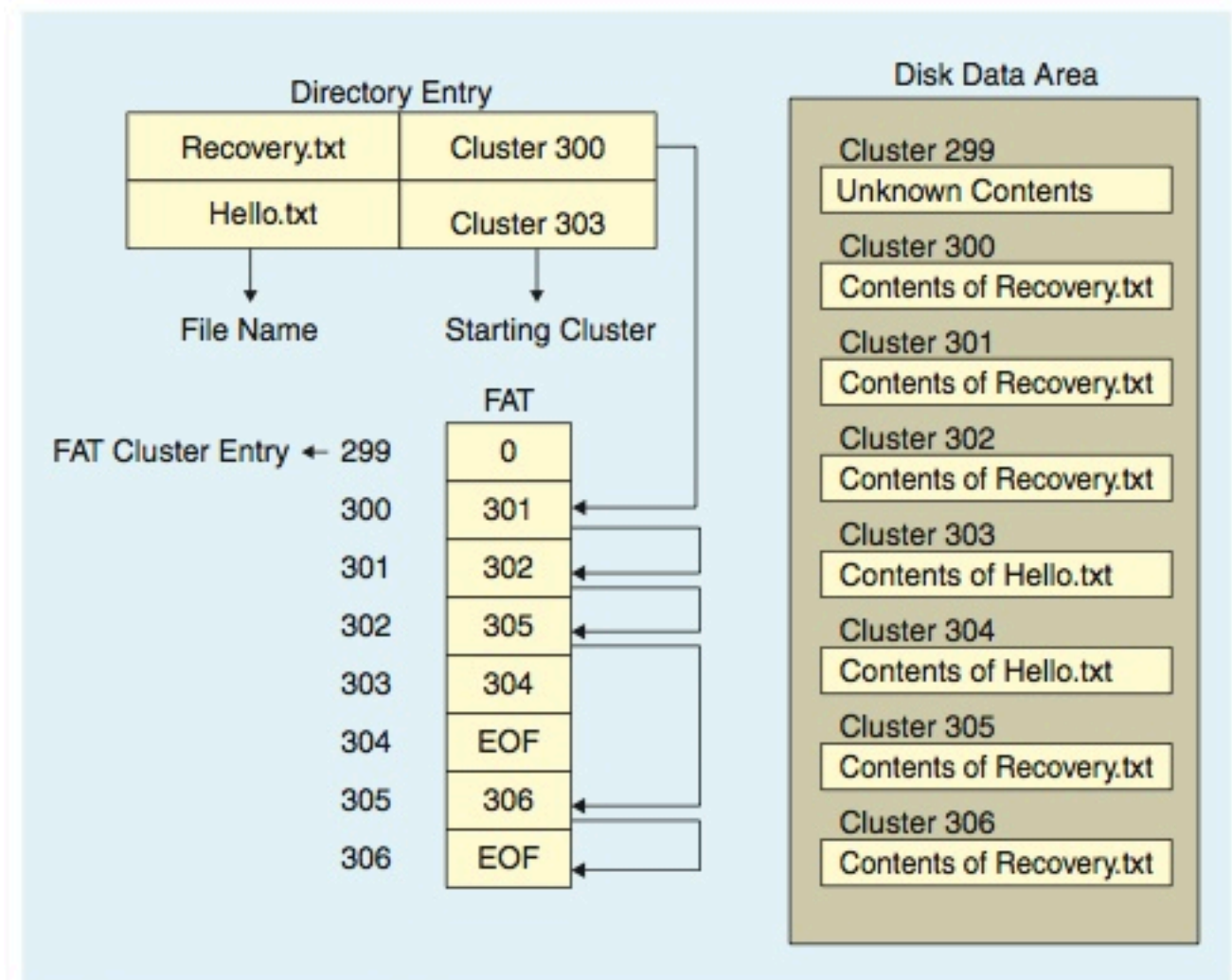
- ▶ Dependente de estruturas do sistema de arquivos, ex.: tabelas de arquivos
  - Sistemas de arquivos costumam apenas “marcar” uma entrada como removida
- ▶ Permite rápida recuperação dos arquivos enquanto estiverem presentes na estrutura
- ▶ Evita buscas em áreas não alocadas do disco



# Recuperação Tradicional

## Inserção - FAT32

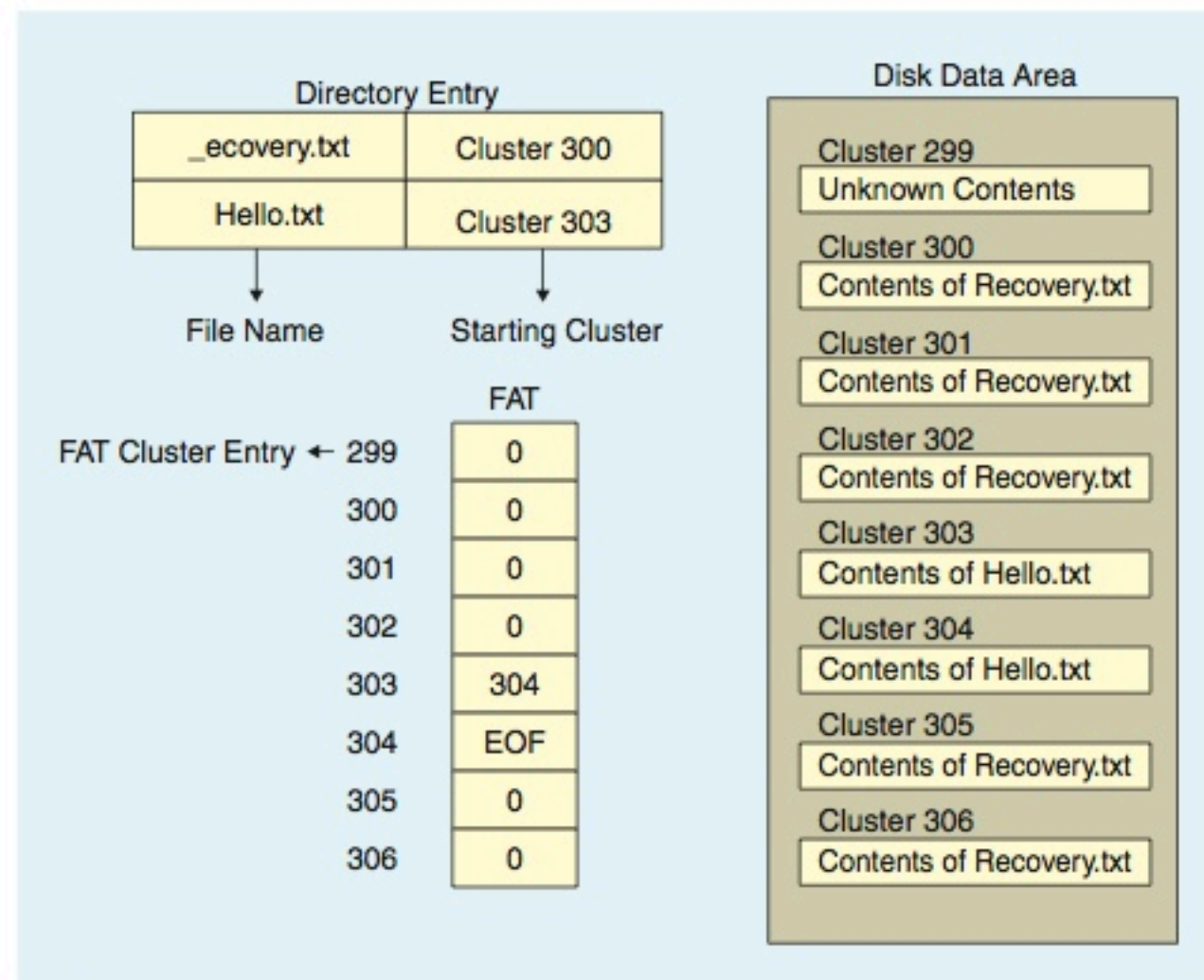
- \* FAT 32 (tabela de alocação de arquivos)
- \* Limitada a 4GB
- \* NTFS veio para superar esse problema
- \* Usa Árvore B para armazenar a informação (não conteúdo) dos arquivos



[FIG1] FAT file allocation for file called recovery.txt, which is stored in clusters 300, 301, 302, 305, and 306. File hello.txt is stored in clusters 303 and 304.

# Recuperação Tradicional

## Deleção - FAT32



[FIG2] FAT file deletion for file called `recovery.txt`, which was stored in clusters 300, 301, 302, 305, and 306. FAT indexes representing the clusters were set to 0 to indicate that they are now available for use. File `hello.txt` was not deleted.

# Algumas Descobertas

- ▶ Em uma análise com 350 HDs (FAT, NTFS, UFS), descobriu-se
  - ▶ Fragmentação é baixa mas existe
  - ▶ Mais alta para arquivos de usuário (MSOffice, e-mail, JPEG).
  - ▶ MS Word = 16%
  - ▶ AVI = 22%
  - ▶ MS-Outlook PSTs = 58%

# Algumas Descobertas

- ▶ Amiga Smart File System busca mover um arquivo inteiro em caso de edição
- ▶ Unix File System (UFS) busca prever possíveis extensões deixando certos clusters disponíveis contíguos a um arquivo
- ▶ XFS e ZFS utilizam escrita tardia até um flush do sistema operacional

# Algumas Descobertas

- ▶ SSDs tendem a aumentar a fragmentação independente do sistema de arquivos devido às técnicas de *wear-leveling*
- ▶ Se o controlador for comprometido, apenas abordagens de file carving poderiam ser utilizadas e não as abordagens tradicionais.
- ▶ Em alguns casos, o próprio sistema de arquivos pode forçar a fragmentação (UFS faz isso para arquivos grandes ou quando o arquivo tem um número ímpar de clusters).

# **File Carving**

# File Carving: Regras gerais

- ▶ Não utiliza a estrutura ou informações do sistema de arquivos de forma direta
- ▶ Busca em blocos não alocados somente quando existe alguma confiança na informação do sistema de arquivos
- ▶ Costumam identificar arquivos muito comuns através de hashes (MD5) e palavras chave

# **File Carving por Estrutura**



# File Carving: Recuperação baseada em estrutura

- ▶ Busca pelos “numeros mágicos” (ie. sequência de bytes em pontos conhecidos)
  - Cabeçalho e rodapé, ou
  - Cabeçalho + tamanho do arquivo
- ▶ Técnicas mais avançadas também utilizam o conteúdo do arquivo
- ▶ O arquivo é formado por todos os blocos que se encontram entre o setor do cabeçalho e o setor do final

# Exemplo: Recuperação baseada em estrutura

A1	A2	B1	B2	B3	C2	C1	C3	A3
----	----	----	----	----	----	----	----	----

Arquivo A: A1+A2+B1+B2+B3+C2+C1+C3+A3

A1	A2	B1	B2	B3	C2	C1	C3	A3
----	----	----	----	----	----	----	----	----

Arquivo B: B1+B2+B3

A1	A2	B1	B2	B3	C2	C1	C3	A3
----	----	----	----	----	----	----	----	----

Arquivo C: C1+C3

# **File Carving por Teoria dos Grafos**

# Recuperação baseada em grafos

- ▶ Tenta resolver o problema de recuperação baseada em estrutura para arquivos fragmentados
- ▶ Os blocos representam os vértices
- ▶ Arestas representam a verossimilhança entre blocos através de um peso
- ▶ Como definir a verossimilhança?

# Grafos: Caminho Hamiltoniano

- ▶ Técnica apresentada por *Shanmugasundaram et al.*
- ▶ Computa a permutação de um conjunto de  $n$  blocos pertencentes ao arquivo  $A$  que representa a estrutura original de  $A$ 
  - Os pesos entre dois blocos representam a probabilidade de serem adjacentes
  - A permutação correta é provavelmente a que maximiza a soma dos pesos
- ▶ O conjunto de todos os pesos forma uma matriz de adjacência de um grafo completo de  $n$  vértices.
  - A sequência correta é um caminho no Grafo.

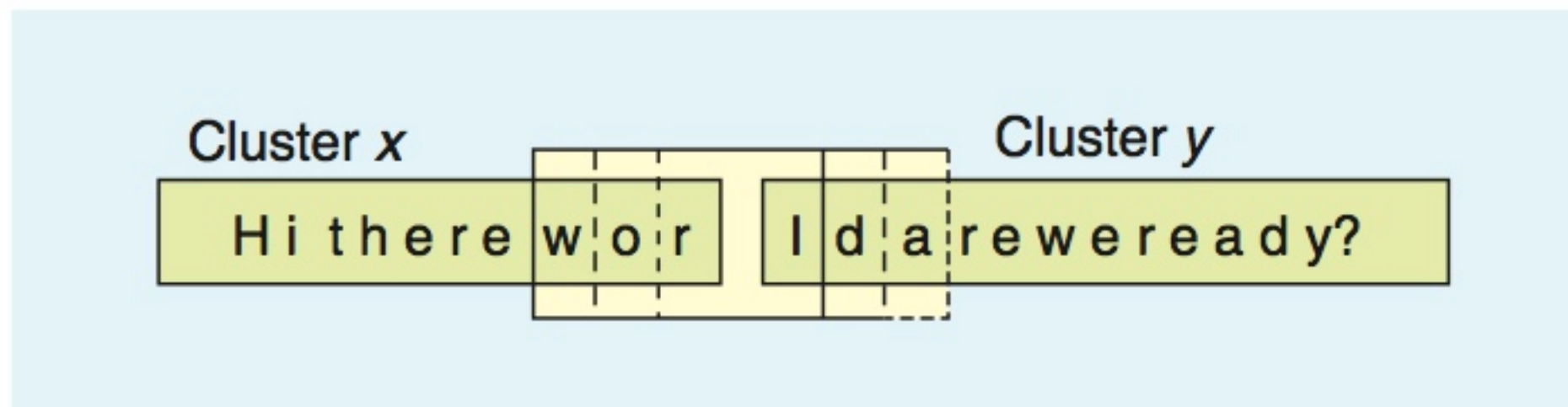
# Grafos: Caminho Hamiltoniano

- ▶ Podemos ver isso como um problema em grafos
- ▶ Todos os pesos candidatos  $W$  formam uma matriz de adjacência de um grafo completo com  $n$  vértices (vértice  $x$  representa o cluster  $bx$ )
- ▶ O peso de aresta  $W(x,y)$  representa a probabilidade do cluster  $by$  seguir o cluster  $bx$ .
- ▶ A permutação correta que resolve esse problema será o caminho no grafo que passa por todos os vértices e maximiza a soma dos pesos candidatos ao longo do caminho.
- ▶ Achar esse caminho é equivalente a achar o caminho Hamiltoniano em um grafo completo.

# Grafos: Caminho Hamiltoniano

- ▶ A pergunta é, então, como determinar o peso entre dois blocos?
  - Prediction by partial matching (PPM) para textos (*Kulesh et al.*)
  - Comparação de bordas entre blocos para imagens (*Pal et al.*)

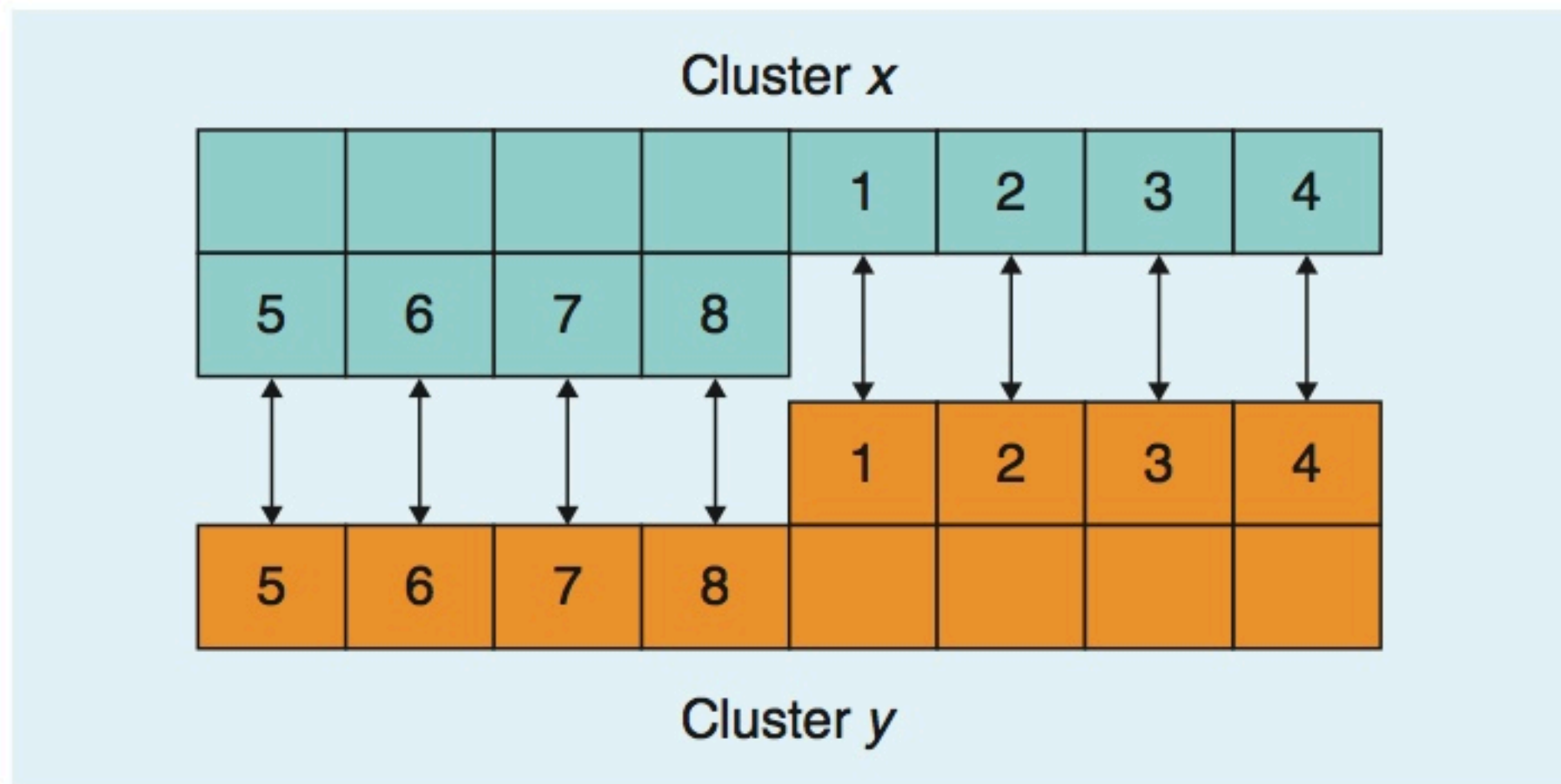
# Prediction by partial matching (PPM)



**[FIG5]** Incorrect sequential recovery shown for an image from DFRWS 2006 that is fragmented.



# Comparação de bordas entre blocos para imagens

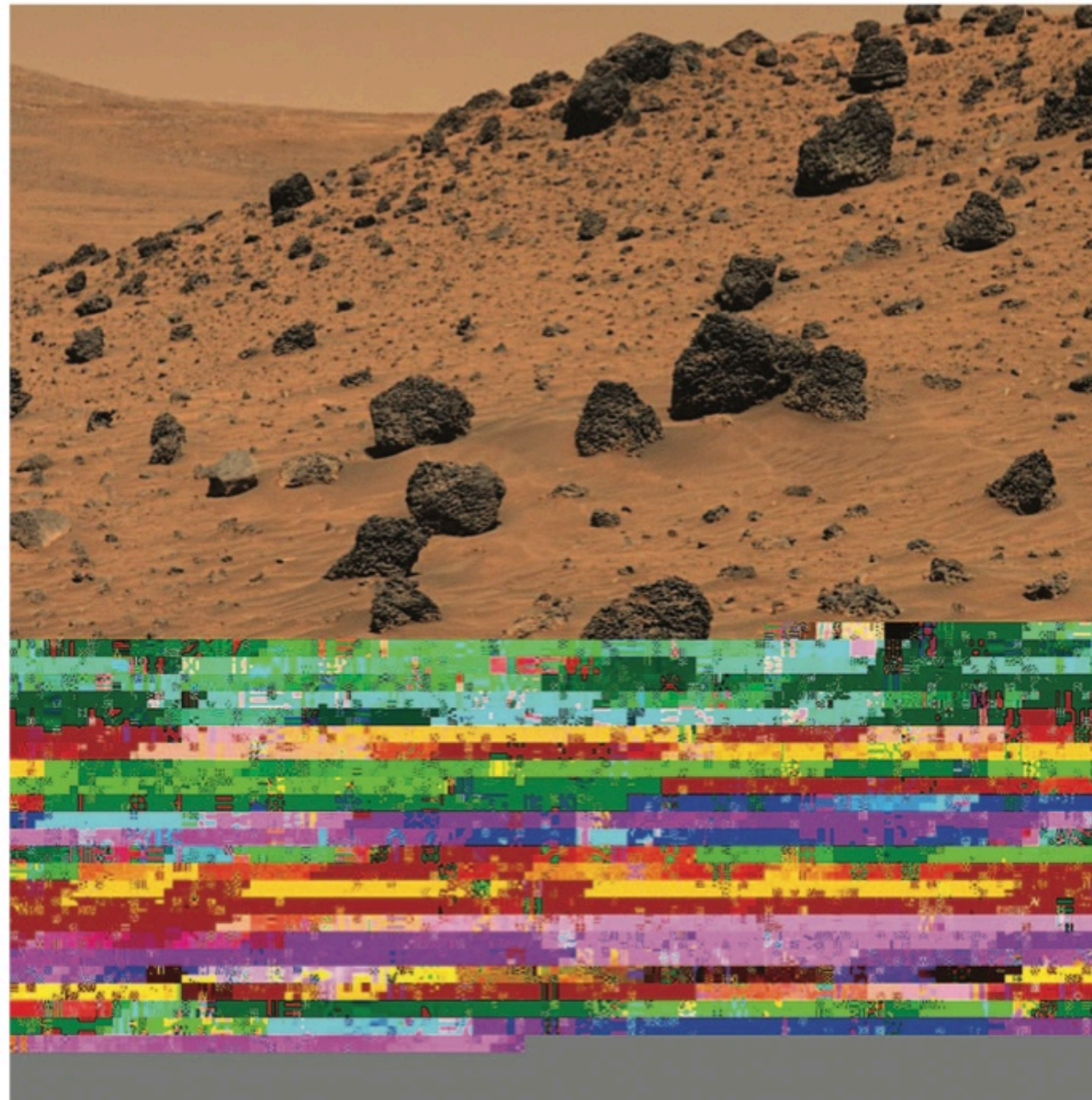


**[FIG6]** Image weighting technique where the sum of difference between each same numbered pixel in cluster X is compared with cluster Y.

# Grafos: Caminho Hamiltoniano

- ▶ Problemas?
  - ▶ Não considera que em sistemas reais vários arquivos podem estar fragmentados ao mesmo tempo.
  - ▶ Estatísticas de múltiplos arquivos poderiam ajudar no processo

# Grafos: Caminho Hamiltoniano



**[FIG4]** Incorrect sequential recovery shown for a fragmented image from DFRWS 2006.

# Grafos: k-Vertex Disjoint Path

- ▶ Refinamento do método do caminho Hamiltoniano por *Pal et al.*
  - Em casos reais, múltiplos arquivos estão fragmentados juntos
  - Utiliza a estatística destes múltiplos arquivos
- ▶ Cada vértice representa um bloco

# Grafos: k-Vertex Disjoint Path

- ▶ k arquivos identificados pelo cabeçalho
  - Existem apenas k caminhos disjuntos, pois (usualmente) cada bloco pertence unicamente à um arquivo
- ▶ É um problema NP difícil
- ▶ Foram propostos algoritmos de caminho único (*UP – unique path*)



# Algoritmos Unique Path (UP)

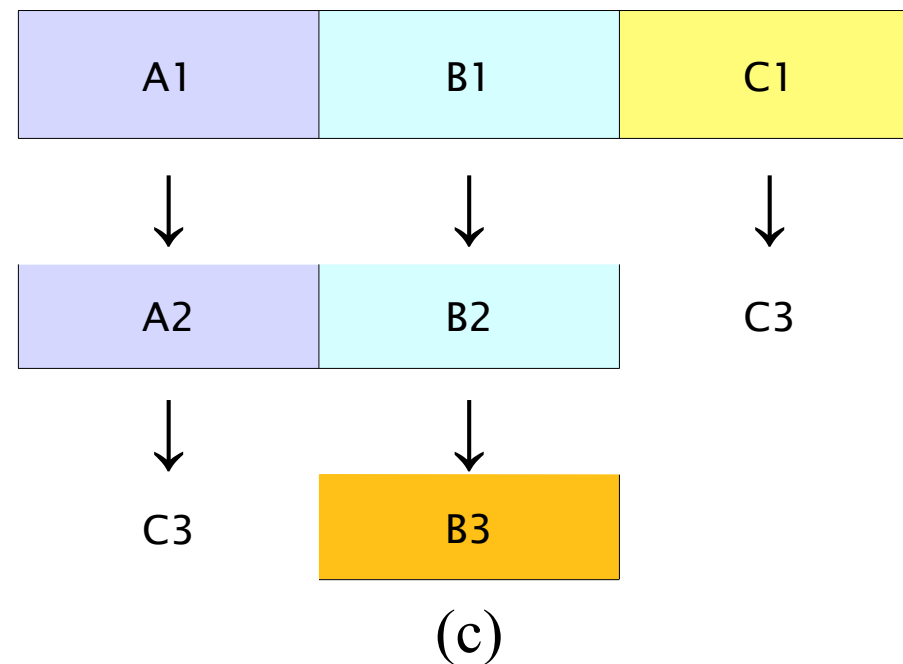
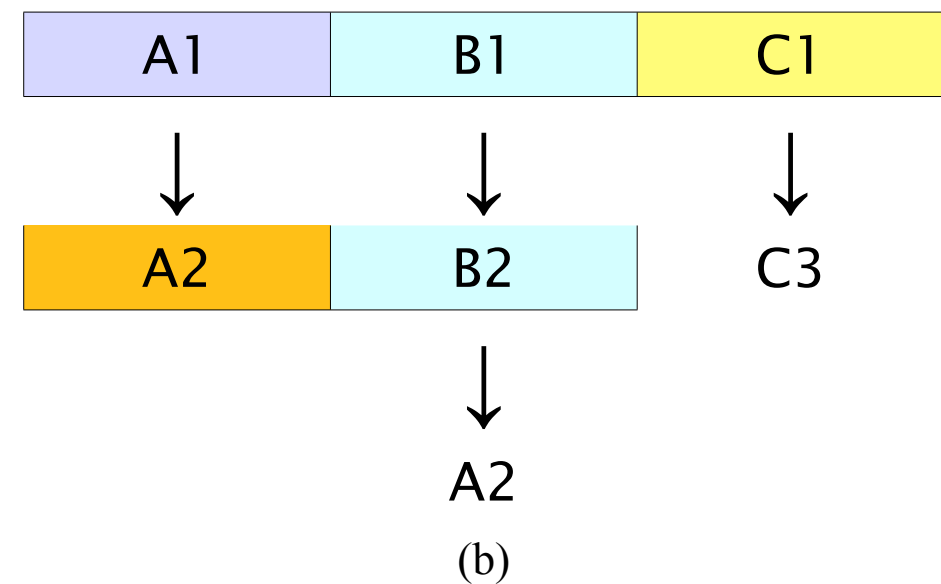
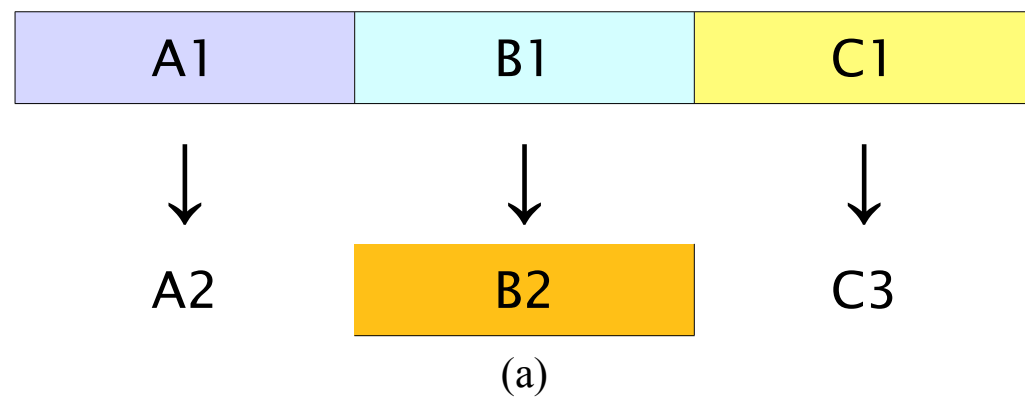
- ▶ Realista: cada bloco costuma pertencer a um único arquivo
- ▶ Erros se propagam em cascata
  - Um bloco incorreto leva à reconstrução errada de 2 arquivos

# File Carving: PUP

## PUP: Parallel Unique Path

1. Começa com um conjunto **S** com  $k$  Headers ( $s_1, s_2, \dots, s_k$ ), referentes a  $k$  arquivos.
2. Encontra um conjunto **T** com  $k$  setores, onde  $t_i$  é a melhor correspondência ao  $s_i$ .  
Seleciona o  $t_i$  com correspondência mais alta dentre todos.
  - i. Adiciona o  $t_i$  ao caminho do  $i$ -ésimo arquivo
  - ii. Substitui o setor corrente em **S** para o  $i$ -ésimo elemento ( $s_i = t_i$ )
  - iii. Encontra um novo conjunto **T** das melhores correspondencias
  - iv. Seleciona o elemento com a melhor correspondencia
  - v. Repete (i) até todos os arquivos estarem completos.

# Exemplo: PUP





# File Carving: SPF

## SPF: Shortest Path First

1. Para cada uma das imagens a serem reconstruídas:
  - i. A partir do conjunto de todos os blocos disponíveis, reconstrói o caminho da imagem e calcula o custo médio do caminho
2. Encontra dentre todos os caminhos aquele com o menor custo e reconstrói essa imagem
3. Remove dos outros caminhos os blocos utilizados na reconstrução do passo (2)
4. Repete o passo (1) até que todas as imagens sejam reconstruídas

# File Carving: PUP vs. SPF

- ▶ Reconstrução de até 88% dos arquivos contra 83% do PUP
- ▶ Tem performance e escalabilidade muito mais baixas que o PUP
- ▶ Os pesos das arestas são pré-computados para facilitar a busca, porém este passo tem complexidade  $O(n^2 \log n)$
- ▶ Discos modernos contêm milhões de clusters e pre-computar os pesos podem ser infactível.

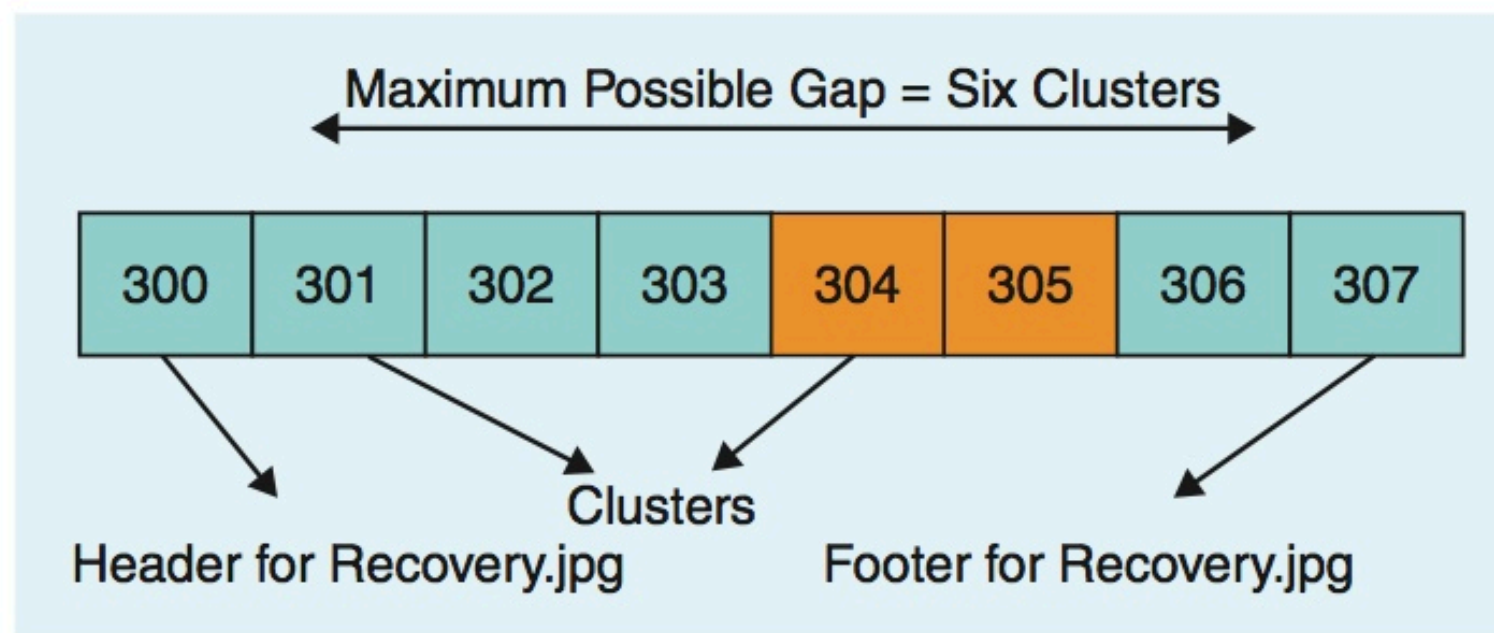
# BitFragment Gap Carving

# BitFragment Gap Carving

- ▶ Fast Object Validation para arquivos com cabeçalho e rodapé
- ▶ Arquivos precisam ser decodificáveis (JPEG, MPEG, ZIP, etc.)
- ▶ Um validador indicará se uma sequência viola a estrutura de um tipo específico de arquivo
- ▶ Por exemplo, PNG usa códigos corretores de erro ao final da imagem
- ▶ Textos planos e BMPs não podem ser recuperados dessa forma

# BitFragment Gap Carving

- ▶ Bitfragment Gap Carving (BGC) recupera arquivos por busca exaustiva do gap entre uma sequência e outra sempre validando o que há no meio



**[FIG8]** Simplified example of BGC algorithm, where for recovery a gap size of two must be chosen and the clusters 300–303 and clusters 306–307 result in successful recovery of file.

# BitFragment Gap Carving

- ▶ Considere  $bh$  como o header cluster,  $bf$  o ponto de fragmentação,  $bs$  o início do cluster com o footer e  $bz$  o footer
- ▶ Para cada tamanho de gap  $g$  começando de 1, todas as combinações de  $bf$  e  $bs$  são designadas de modo que haja exatamente  $g$  clusters entre eles ( $s - f = g$ )
- ▶ **Desvantagens:**
  - ▶ Essa técnica não escala para maiores *gaps*
  - ▶ Funciona para arquivos com dois fragmentos
  - ▶ Só funciona para arquivos que podem ser validados
  - ▶ Validação coerente não significa recuperação correta de arquivo

# BitFragment Gap Carving



**[FIG9]** Fully validated but incorrect JPEG image from the DFRWS 2007 test set.

# Smart File Carving

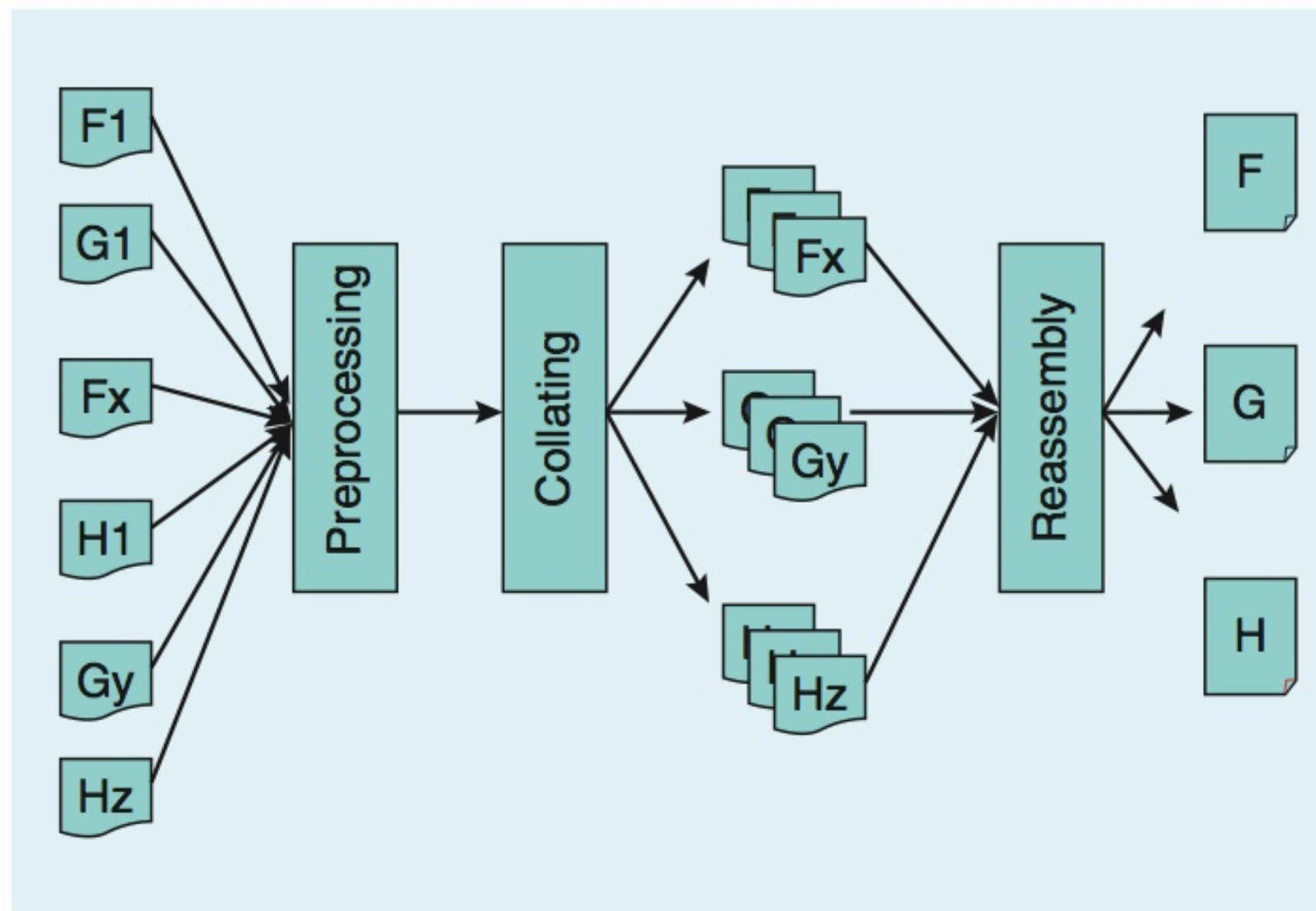


# Smart Carving

Proposto por *Pal et al.*

- ▶ Visa resolver problemas de escalabilidade
- ▶ Leva em consideração o comportamento típico da fragmentação em discos
- ▶ Etapas:
  - Pré processamento
  - Comparação/classificação
  - Reconstrução

# Smart Carving



**[FIG10]** The three components required in a file carver handling fragmented files are (a) preprocessing, (b) collation, and (c) reassembly.

# Smart Carving: Pré-processamento

- ▶ Aplicada em dados com compressão ou encriptação
- ▶ Opcionalmente pode remover os blocos reconhecidamente alocados

# Smart Carving: Comparação e Classificação

- ▶ Classifica os blocos pelo tipo de arquivo
  - Palavras chave ou padrões (e.g., <HTML>, <IMG>)
  - Frequência de caracteres ASCII
  - Entropia
  - “Impressão digital” de arquivos (e.g., histograma de bytes dos arquivos)

# Smart Carving: “Impressão digital”

- ▶ *McDaniel and Heydari* propuseram 3 algoritmos:
  - Frequência de distribuição de bytes (BFD): média dos histogramas de diversos exemplos e correlação dos bytes
  - Frequência de distribuição de correlação cruzada (BDC): correlação entre os bytes
  - Inclusão de cabeçalho e rodapé
- ▶ Baixa acurácia: 30% (BFD), 45% (BFC) e 95% com os cabeçalhos e rodapés
- ▶ Não funciona bem para blocos

# Smart Carving: “Impressão digital”

Proposto por *Wang and Stolfo*

- ▶ Utiliza um conjunto de modelos BFD e desvio padrão
- ▶ Melhor acurácia: entre 75% e 100%
- ▶ Acurácia decresce com o número de bytes disponíveis

# Smart Carving: “Impressão digital”

*Karresand et al.* propuseram o método Oscar

- ▶ Utiliza um modelo de centróide baseado na média e desvio padrão de cada byte
  - Obteve uma acurácia de 97%
- ▶ Melhorado ao incorporar uma medida do ordenamento dos bytes utilizando a diferença absoluta entre bytes adjacentes
  - Acurácia de 99% para JPEG

# Smart Carving: Reconstrução

- ▶ Visa localizar o ponto de fragmentação de um arquivo
- ▶ Estudos anteriores mostraram que, em geral, os arquivos raramente se fragmentam em mais de três fragmentos
- ▶ A reconstrução consiste em encontrar o fragmento-base e localizar seu último bloco

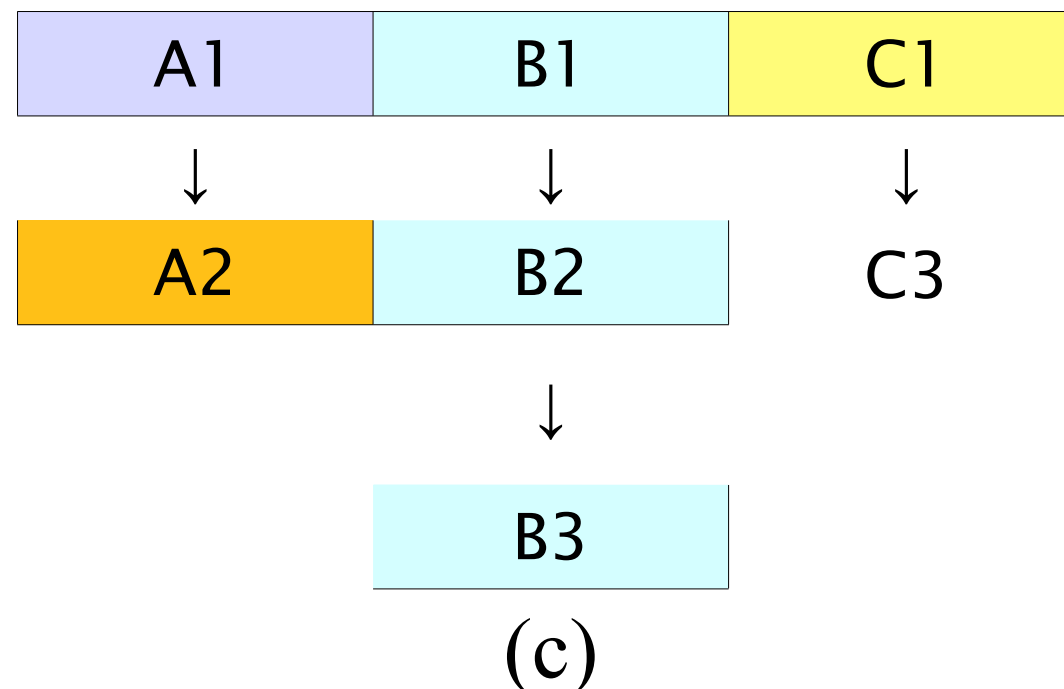
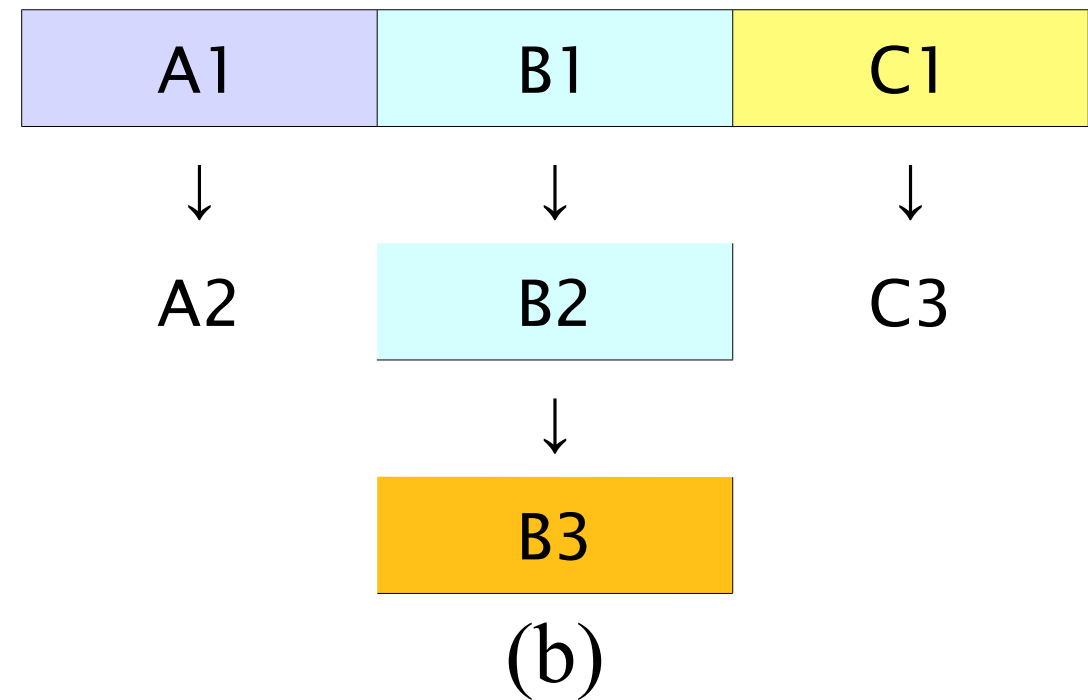
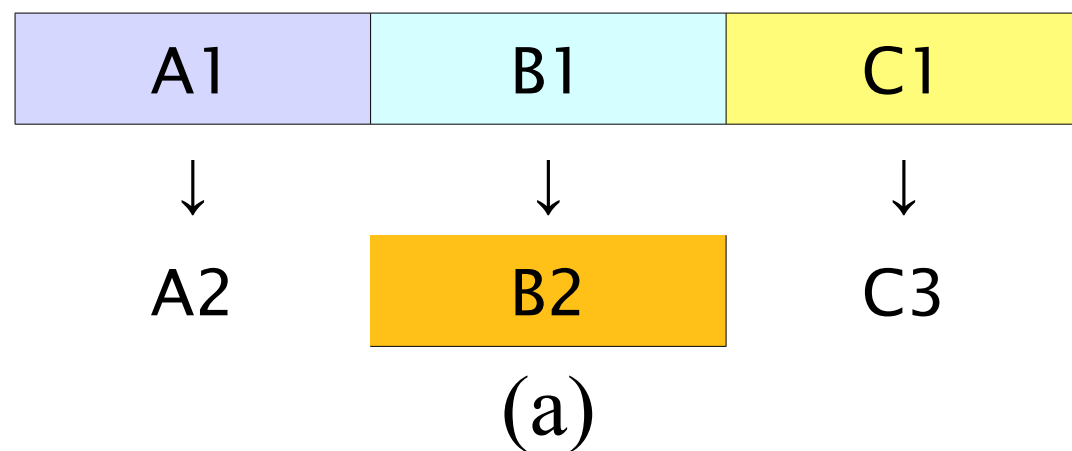


# Smart Carving: SHT-PUP

- ▶ Modificação do PUP por *Pal et al.*
- ▶ SHT: Sequential Hypotesys Testing
- ▶ Cada tipo de arquivo possui uma hipótese
- ▶ Blocos são anexados ao caminho até que a Hipótese seja confirmada ou refutada
- ▶ Só foi implementado para imagens JPEG

1. Começa com um conjunto **S** com  $k$  Headers ( $s_1, s_2, \dots, s_k$ ), referentes a  $k$  arquivos.
2. Encontra um conjunto **T** com  $k$  setores, onde  $t_i$  é a melhor correspondência ao  $s_i$ .  
Seleciona o  $t_i$  com correspondência mais alta dentre todos.
  - i. Adiciona o  $t_i$  ao caminho do  $i$ -ésimo arquivo
  - ii. Substitui o setor corrente em **S** para o  $i$ -ésimo elemento ( $s_i = t_i$ )
  - iii. Analiza sequencialmente os setores imediatamente depois de  $t_i$  até detectar um ponto de fragmentação  $t_f$  ou o arquivo estar completo. Consiste de um teste de hipótese que é executado para cada setor subsequente adicionado ao fragmento. O teste verifica se a sequência completa pertence ou não ao fragmento inteiro. Se o resultado for inconclusivo o próximo setor é adicionado.
  - iv. Substitui o setor corrente em **S** com  $t_f$  ( $s_i = t_i$ )
  - v. Encontra um novo conjunto **T** das melhores correspondências
  - vi. Seleciona o elemento  $t_i$  com a melhor correspondência
  - vii. Repete passo (i) até todos os arquivos estarem completos.

# Exemplo: SHT-PUP



# Conclusões

# Conclusões

Foi feita uma análise abrangente das técnicas existentes para a recuperação de arquivos sem o uso de nenhuma meta-informação do sistema de arquivos.

Uma técnica de recuperação de arquivos de texto e imagens foi apresentada. Esta técnica, apesar de útil, precisa ser ampliada para incorporar recuperação de arquivos de vídeo, áudio, executáveis e outros formatos.

# Referências

# Referências

1. Anandabrata Pal, Husrev T. Sencar, and Nasir Memon. Detecting file fragmentation point using sequential hypothesis testing. Digital Investigation (DIIN), 5(1):S2–S13, September 2008.
2. Anandabrata Pal and Nasir Memon. The evolution of file carving – the benefits and problems of forensics recovery. IEEE Signal Processing Magazine, 26(2):59–71, March 2009.
3. Pal A, Memon N. Automated reassembly of file fragmented images using greedy algorithms. IEEE Transactions on Image processing February 2006:385–93.
4. K. Shanmugasundaram and N. Memon, “Automatic reassembly of document fragments via data compression,” presented at the 2nd Digital Forensics Research Workshop, Syracuse, NY, July 2002
5. A. Pal, K. Shanmugasundaram, and N. Memon, “Reassembling image fragments,” in Proc. ICASSP, Hong Kong, Apr. 2003, vol. 4, pp. IV–732-5.
6. A. Pal and N. Memon, “Automated reassembly of file fragmented images using greedy algorithms,” IEEE Trans. Image Processing, vol. 15, no. 2, pp. 385 – 393, Feb. 2006.
7. A. Pal, T. Sencar, and N. Memon, “Detecting file fragmentation point using sequential hypothesis testing,” Digit. Investig., to be published.
8. M. McDaniel and M. Heydari, “Content based file type detection algorithms,” in Proc. 36th Annu. Hawaii Int. Conf. System Sciences (HICSS’03)—Track 9, IEEE Computer Society, Washington, D.C., 2003, p. 332.1
9. K. Wang, S. Stolfo, “Anomalous payload-based network intrusion detection,” in Recent Advances in Intrusion Detection, ( Lecture Notes in Computer Science), vol. 3224. New York: Springer-Verlag, 2004, pp. 203 –222.
10. M. Karresand and N. Shahmehri, “Oscar file type identification of binary data in disk clusters and RAM pages,” in Proc . IFIP Security and Privacy in Dynamic Environments, vol. 201, 2006, pp. 413 – 424.
11. M. Karresand and N. Shahmehri, “File type identification of data fragments by their binary structure,” in Proc. IEEE Information Assurance Workshop, June 2006, pp. 140 –147.

**NOTA:** os artigos de A. Pal et al. podem ser obtidos em <http://digital-assembly.com/technology/>

---

***Obrigado!***

---