



Engenharia de Computação



Especialização Lato Sensu em Ciência de Dados e Analytics

Soluções em Processamento para Big Data

{ Escalabilidade e Tolerância a Falhas }

Prof. Jairson Rodrigues
jairson.rodrigues@univasf.edu.br

{ arquitetura de referência }

AGENDA

- Big Data: Um Novo Paradigma
- Modelo de Dados para Big Data
- Batch Layer
- Serving Layer
- Speed Layer



{ escalabilidade em sistemas tradicionais }

Exemplo: Web Analytics

Informar a qualquer momento as 100 url's mais visitadas



Column name	Type
id	integer
user_id	integer
url	<u>varchar(255)</u>
<u>pageviews</u>	<u>bigint</u>

{ vamos colocar em produção }

- A solução faz sentido?
 - SIM (pelo menos em um mundo antes de Big Data)
- Sistema no ar, todos felizes

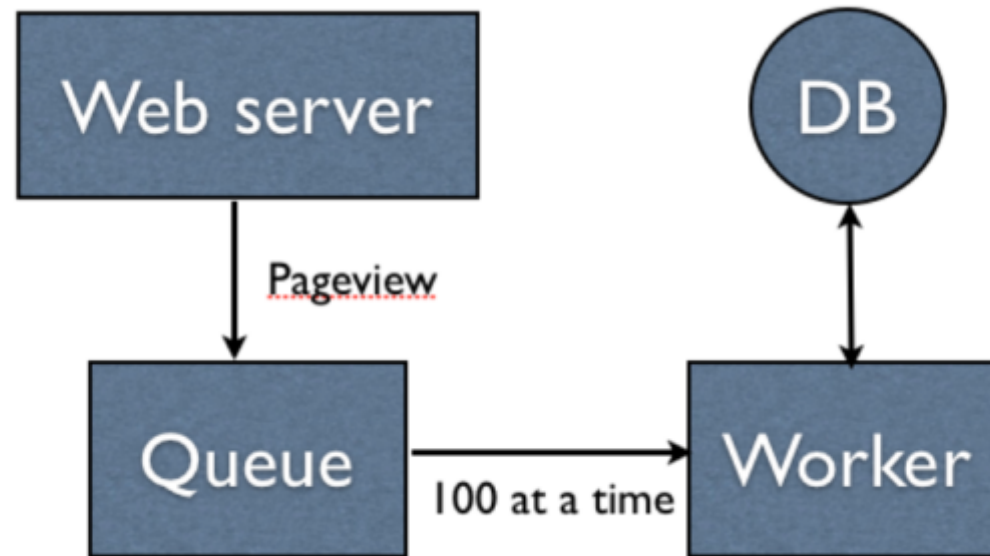
Durante o churrasco de comemoração... no smartphone do gerente do projeto

"Timeout error on inserting to the database."



O Grito – Edvard Munch (1893, óleo sobre tela, 91 x 73 cm)

{ escalabilidade com filas }



Fila para atualizações em segundo plano [1]

{ e mais complexidade com a escalabilidade }

- A aplicação se torna mais popular, um worker não é mais o bastante,
- Mais workers não resolvem a longo prazo
- Solução:
 - Tabelas esparsas ao longo de múltiplos servidores (“particionamento horizontal”)
- Associar cada fragmento a uma chave computando função de hash módulo número fragmentos (DHT)

{ e mais complexidade com a escalabilidade }

- script para mapear dados espalhados em cada fragmento
- perda de dados em tempo real na substituição (worker precisa ser desligado para manutenção)
- top 100 URL's de fragmentos ao invés de top 100 URL's de uma única fonte de dados
- Com o tempo... necessidade de maior fragmentação
- complexidade sempre crescente



{ e sobre tolerância a falhas? }

- Muitos fragmentos -> uma das máquinas pode sair do ar
- Solução
 - Atualizar a consulta e o worker para colocar dados em uma fila separada, a ser descarregada a cada, por exemplo, cinco minutos
 - Replicação com servidores slave

{ e quanto a dados corrompidos? }

- Imagine a inserção de um bug em produção durante as alterações de funcionalidade
 - Incremento de pageviews por dois, acidentalmente
- Você, programador, só nota o bug 5 horas depois
- Seu backup **diário** não resolve o problema

“Antigamente eu gastava meu tempo construindo novas funcionalidades para os clientes. Agora gasto todo o meu tempo lidando com problemas de infraestrutura, tolerância a falhas, escalabilidade e corrupção dos dados”

Adaptado de (Nathan Marz & James Warren, 2014)

{ paradigma Big Data }

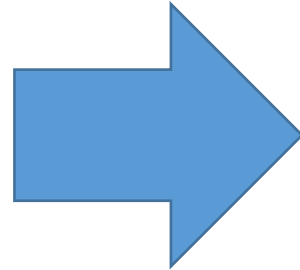
- Volume de dados **incompreensivelmente grande, várias fontes, alta velocidade**
 - Geração de dados de forma intensa e diversa: posts de blog, tweets, interações em rede social, fotos, vídeos, mapas, dados gerados por máquina etc.

“Algumas das formas mais básicas para se lidar com dados em sistemas tradicionais como SGDB’s relacionais (RDBMS) são muito complexas para sistemas Big Data. Uma alternativa mais simples é a abordagem baseada em novos paradigmas, apelidada ‘Arquitetura Lambda’”.

(Nathan Marz & James Warren, 2014)

{ o que queremos? }

- Fragmentação e replicação auto-contidas na arquitetura
- Imutabilidade do dado
- Paradigma NOSQL



- Robustez
- Tolerância a falhas
- Baixa latência de leitura e atualização
- Escalabilidade
- Generalização
- Extensibilidade
- Pouca manutenção

{ alguns princípios }

- Qual a tarefa fundamental de um sistema de dados?
 - No nível mais fundamental: “responder questões baseadas em informações adquiridas no passado”
- Imagine um sistema de dados para redes sociais
 - Qual o nome do usuário?
 - Quantos amigos esse usuário possui?
- Imagine um sistema bancário
 - Qual o saldo de minha conta?
 - Que transações ocorreram recentemente?

{ alguns princípios }

- Sistemas de dados não memorizam e regurgitam informações. Elas são extraídas a partir de dados combinados.
- Saldo de conta pode ser derivado do histórico de transações
- Número de amigos pode ser derivado do histórico de adições e remoções de amigos
- Neste contexto, chamaremos de dado a informação que não é derivada de nenhuma outra.

query = function(all data)

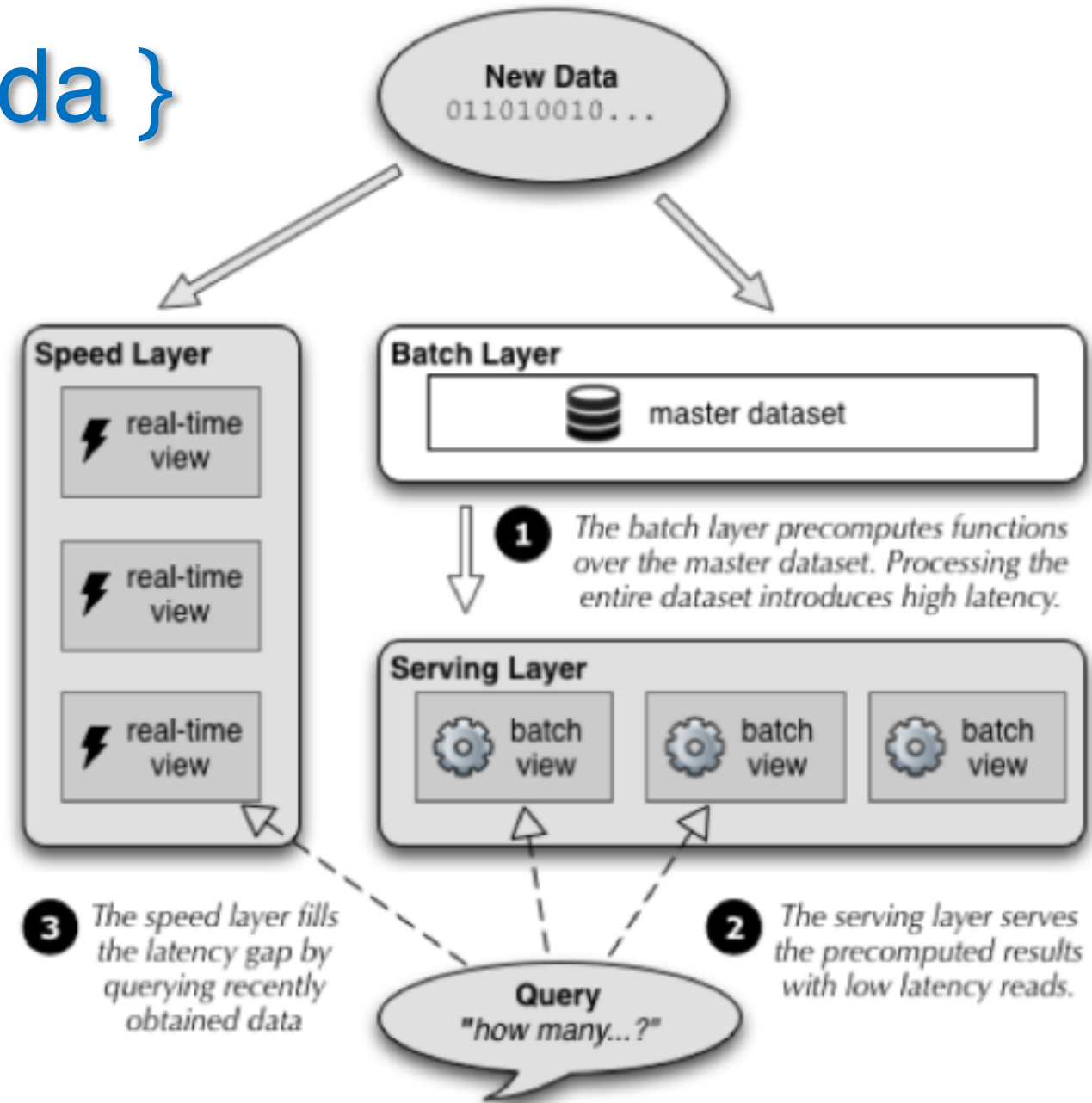
{ arquitetura lambda }

“Abordagem de propósito geral para implementação de funções arbitrárias sob conjunto arbitrário de dados, com retorno de baixa latência”

{ como processar um petabyte? }

- computar funções sobre dados volumosos em tempo real
 - o volume é incompreensível
 - não para de chegar
 - não para de crescer
 - pode vir de várias fontes
- decomposição de funções
 - batch layer
 - serving layer
 - speed layer

{ arquitetura lambda }



{ modelo de dados em big data }

SuperWebAnalytics.com

FaceLambda

Propriedades do Dado

Estado Bruto

Imutabilidade

Perpetuidade

Modelo de Dados Baseado em Fatos



{ superwebanalytics.com }

- Registra bilhões de pageviews por dia
- Consultas possíveis
 - Qual o pageview para cada dia no ano passado?
 - Quantos pageviews houve nas últimas doze horas?
 - Quantos visitantes únicos houve no ano de 2010?
 - Quantos visitantes únicos houve por hora nos últimos três dias?
 - Qual a percentagem de pessoas que visitaram a página inicial do website e não visitaram nenhuma outra página?
- E, principalmente, uma outra pergunta qualquer que o projetista não tenha pensado antes

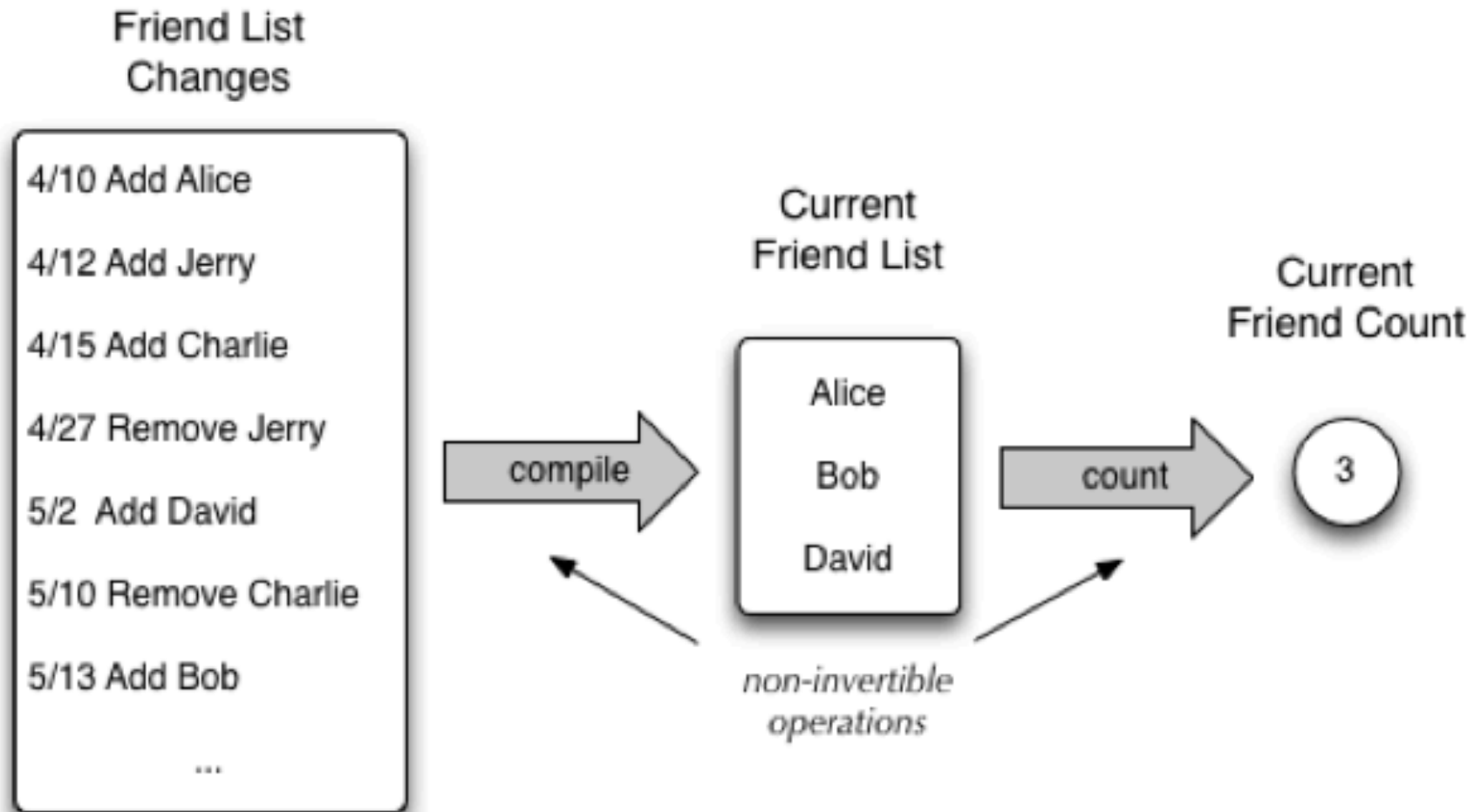
{ conjunto mestre de dados }

- Deve ser absolutamente livre de corrupção de dados
- Armazenamento físico com tolerância
- Deve permitir o processamento em paralelo
- Deve continuar funcionando mesmo na ocorrência de falhas

{ facelambda ⁽¹⁾ }

- Suponha que você vai desenvolver uma nova e grande rede social, **FaceLambda**
- Um novo usuário (Facebuquisson) deseja convidar amigos e família
- Que informações devem ser armazenadas?
 - A sequência de adds e removes de amigos de Facebuquisson?
 - A lista corrente de amigos de Facebuquisson?
 - O número atual de amigos de Facebuquisson?

{ modelo de dados }



{ propriedades-chave do dado }

- Dados em estado **bruto**
- **Imutabilidade** dos dados
- **Perpetuidade** dos dados ou “eterna verdade dos dados”
- Conceitos que podem ser confusos para os estudantes e profissionais habituados com o paradigma relacional no qual dados são inseridos e atualizados

{ dados em estado bruto }

- Sistemas de dados respondem questões sobre dados adquiridos no passado
- Sistemas Big Data devem estar preparados para responder quantas questões for possível
- Quanto mais bruto o dado, mais informações podem ser derivadas do mesmo

{ dados em estado bruto }

Company	Symbol	Previous	Open	High	Low	Close	Net
Google	GOOG	564.68	567.70	573.99	566.02	569.30	+4.62
Apple	AAPL	572.02	575.00	576.74	571.92	574.50	+2.48
Amazon	AMZN	225.61	225.01	227.50	223.30	225.62	+0.01

Dado em estado bruto *versus* dado condensado




{ imutabilidade do dado }

- Não há atualização ou remoção de dados (2)
- Permite-se apenas inserção de dados
- Vantagens
 - Tolerância a falhas humanas
 - Simplicidade

(2) Há raros casos em que dados podem ser removidos, diante do paradigma em questão.
É aceitável, por exemplo, deletar dados de menor valor (garbage collection) ou por exigências legais.

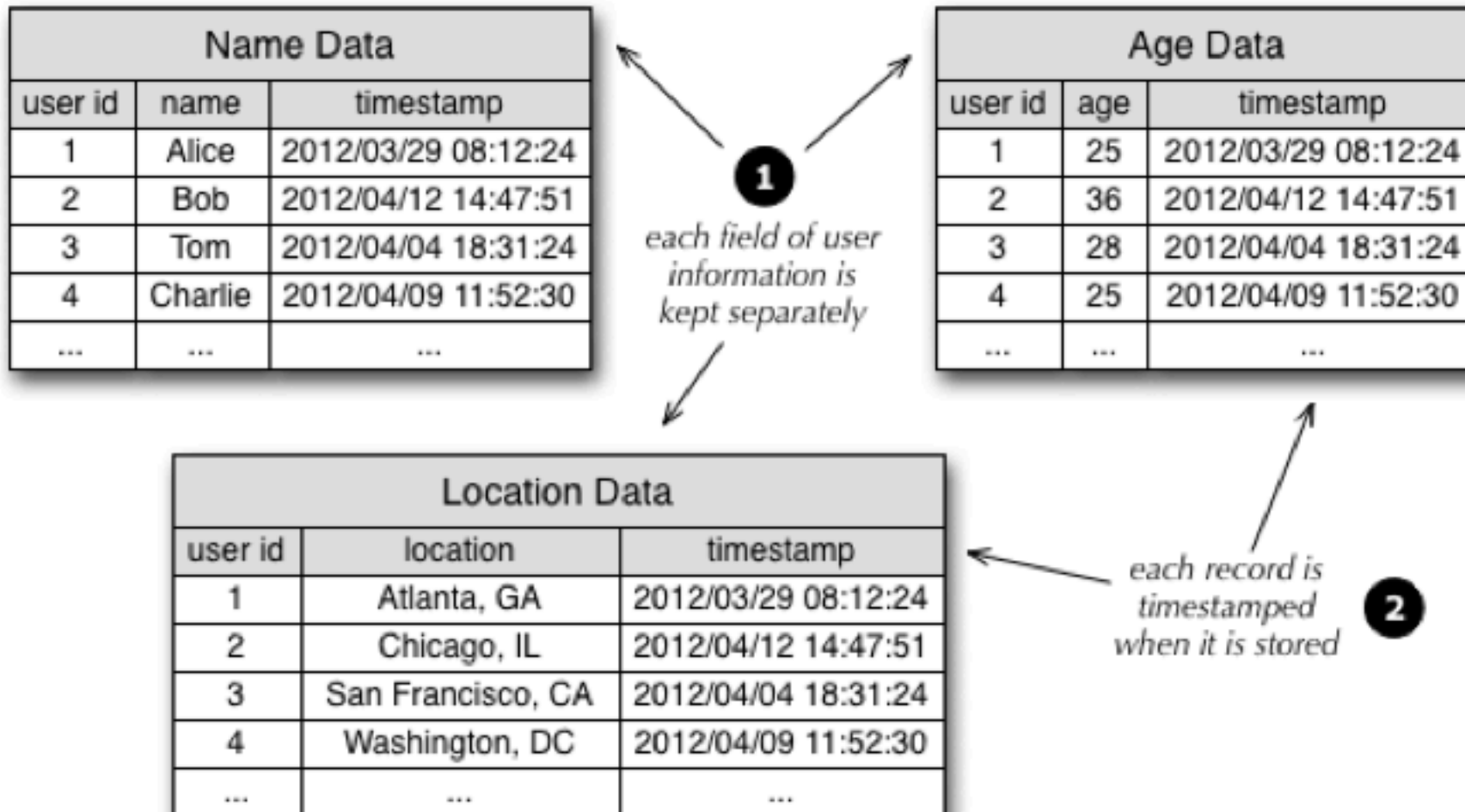
{ imutabilidade do dado - ex: dados mutáveis }

User Information					
id	name	age	gender	employer	location
1	Alice	25	female	Apple	Atlanta, GA
2	Bob	36	male	SAS	Chicago, IL
3	Tom	28	male	Google	San Francisco, CA
4	Charlie	25	male	Microsoft	Washington, DC
...



should Tom move to a different city, this value would be overwritten

{ imutabilidade do dado - ex: dados imutáveis}



{ imutabilidade do dado - ex: dados imutáveis}

Location Data		
user id	location	timestamp
1	Atlanta, GA	2012/03/29 08:12:24
2	Chicago, IL	2012/04/12 14:47:51
3	San Francisco, CA	2012/04/04 18:31:24
4	Washington, DC	2012/04/09 11:52:30
3	Los Angeles, CA	2012/06/17 20:09:48
...

1

the initial information provided by Tom (user id 3), timestamped when he first joined FaceSpace

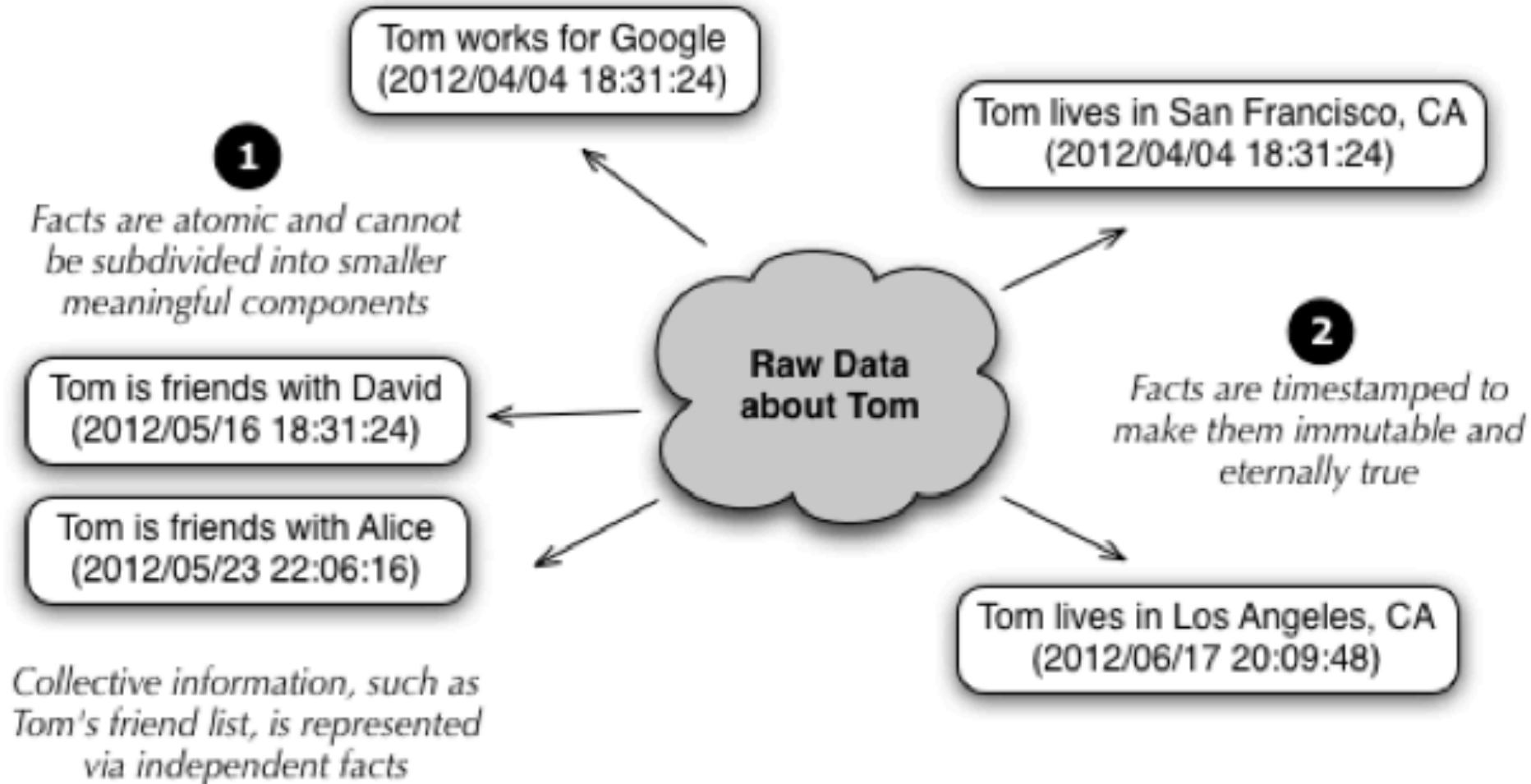
2

when Tom later moves to a new location, we add an additional record timestamped by when we received the new data

{ perpetuidade do dado }

- Uma consequência-chave da imutabilidade dos dados é que cada pedaço de dado, marcado com um timestamp, é verdadeiro para sempre
- Exemplo:
 - O Brasil nunca foi República até 15 de Novembro de 1889
 - Este dado é eternamente verdadeiro
 - O Brasil ter virado República não altera a verdade do dado
 - Se o Brasil virar Monarquia, não alterará a verdade do dado
 - ...

{ representação baseada em fatos }



{ representação baseada em fatos - benefícios }

- “Consultabilidade” do conjunto de dados
 - Dados não são removidos ou alterados
 - É possível **reconstruir o estado do mundo** em qualquer dado momento específico no tempo
- Tolerância a falhas humanas
- Manipulação de informações parciais
 - Fatos ausentes equivalem a NULL
- Consulta e armazenamento em camadas diferentes (4)

(4) Em parte devido à Arquitetura Lambda

{ batch layer }

Conceitos

Armazenamento

Hadoop Distributed Filesystem

Atualização de Batch Views

Algoritmos Recomputacionais

Algoritmos Incrementais

Paradigma MapReduce

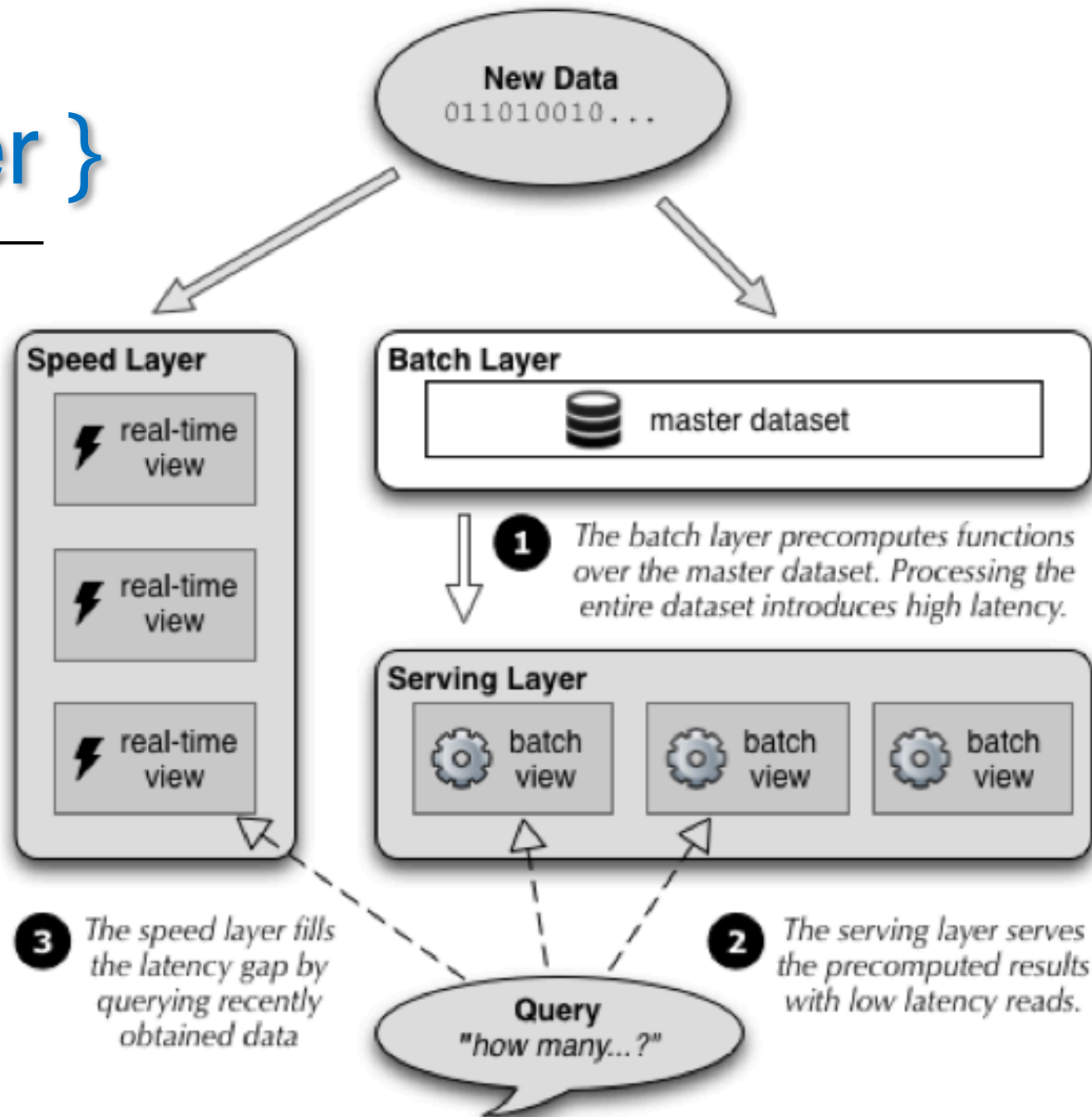


{ batch layer }

query = function(all data)

- Conjunto mestre de dados muito volumoso
- A Batch Layer pré-computa o conjunto de dados em batch views para resolução em baixa latência

{ batch layer }



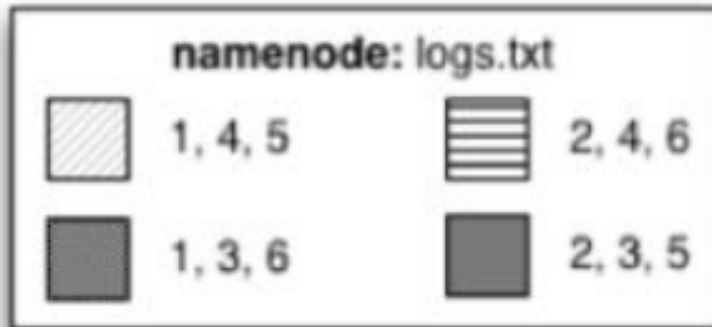
{ conjunto mestre de dados - armazenamento }

- Otimizado para um enorme e **constantemente crescente** conjunto de dados
- Princípios para a Batch Layer
 - Operação “write” muito simples
 - Acesso randômico a pedaços individuais de dados não é necessário
 - Operação “read” eficiente ao ler muitos dados de uma vez
 - Paradigma “**escreva uma vez e proceda com leituras em massa**”

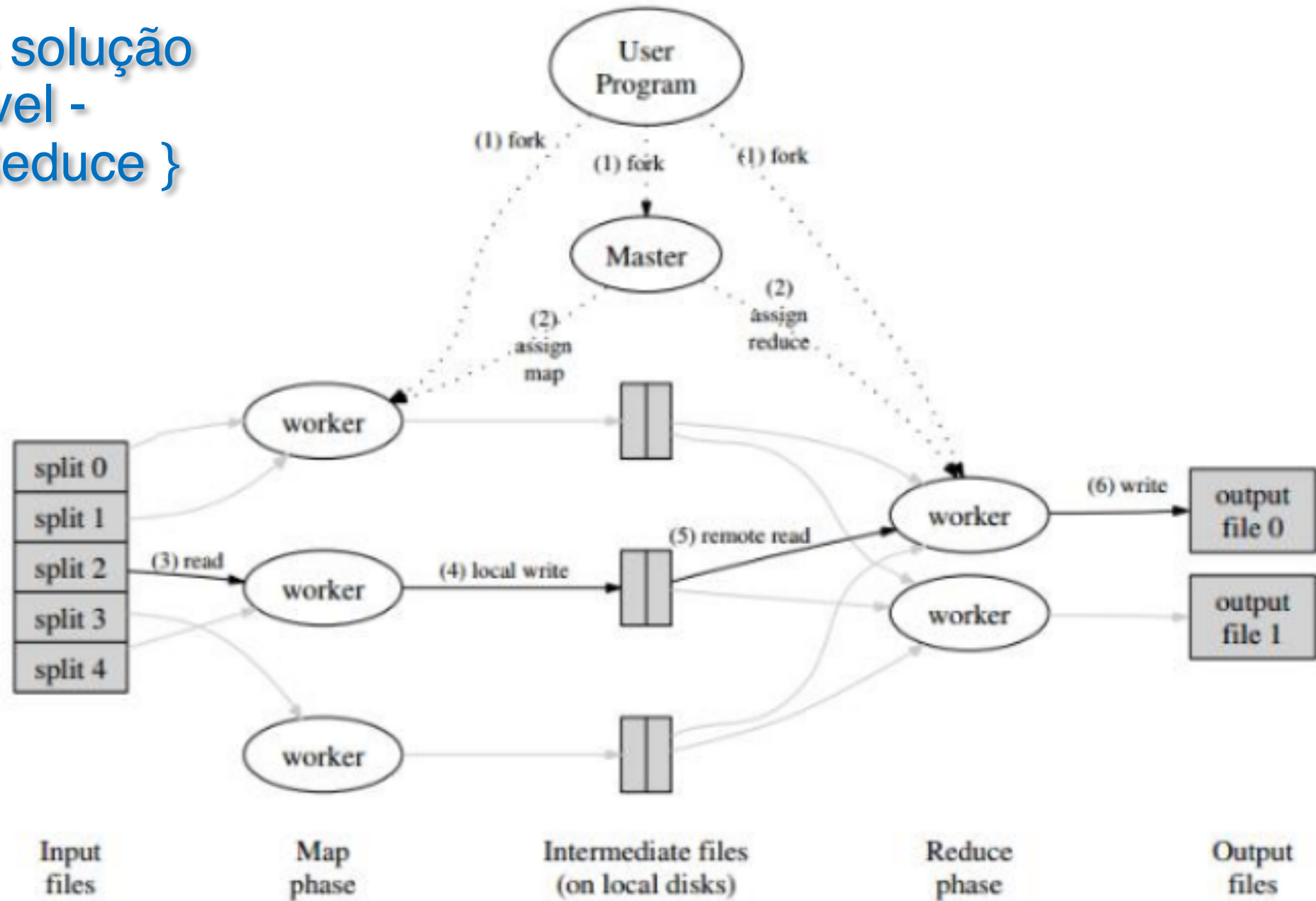
{ conjunto mestre de dados - principais requisitos }

- Escrita
 - Inserção eficiente de novos dados (append)
 - Escalabilidade de armazenamento
- Leitura
 - Suporte para leitores em processamento paralelo
- Leitura/Escrita
 - Equilíbrio entre performance e custo

{ uma solução possível - HDFS }

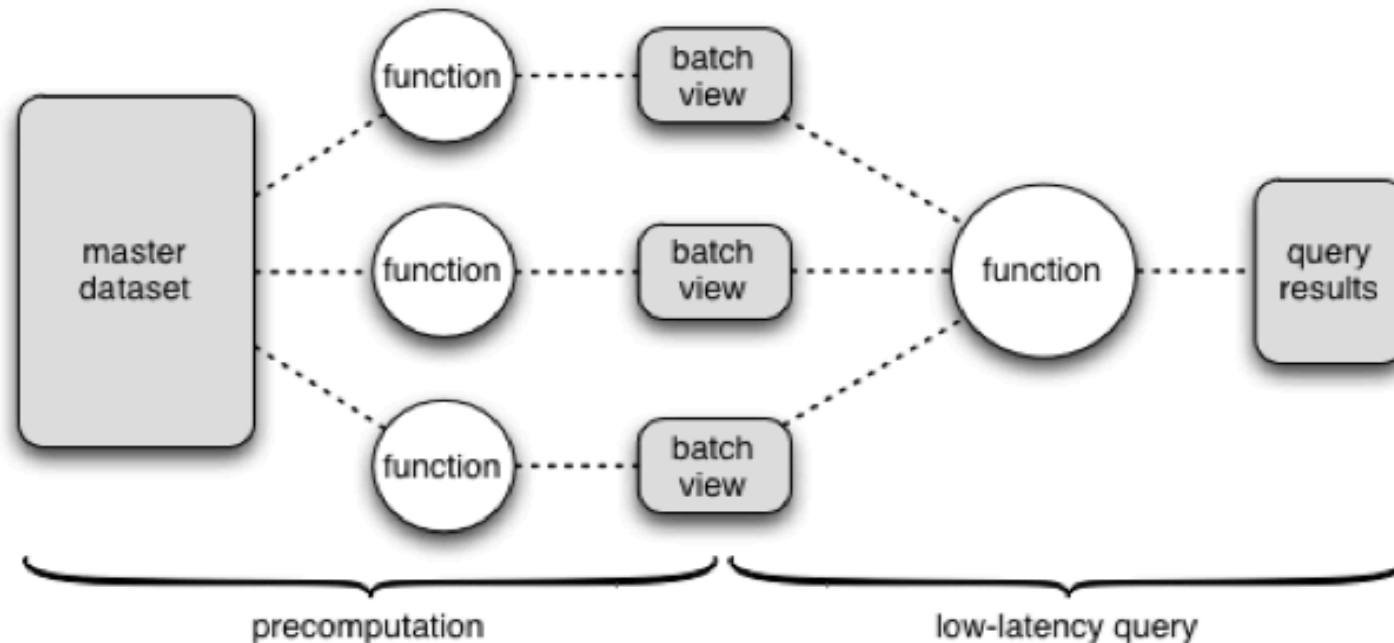


{ uma solução possível - MapReduce }



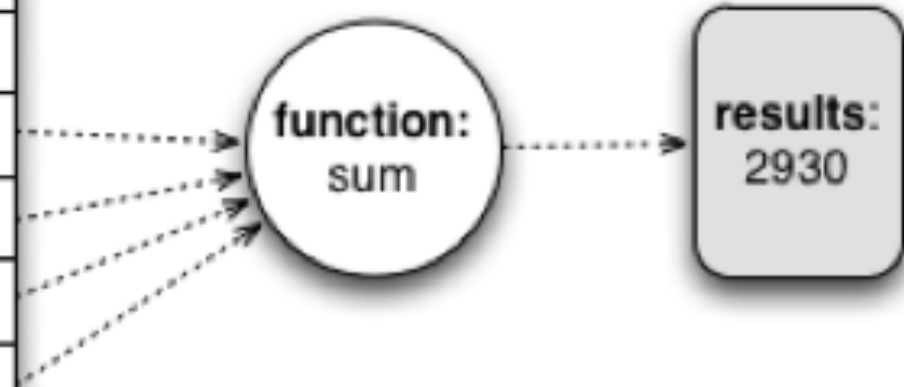
{ exemplo - pageviews por hora }

- Total para cada domínio de horas, para um dado ano (potencialmente **centenas de milhões** de possibilidades)
 - Pré-computar um resultado intermediário e usá-los para completar consultas



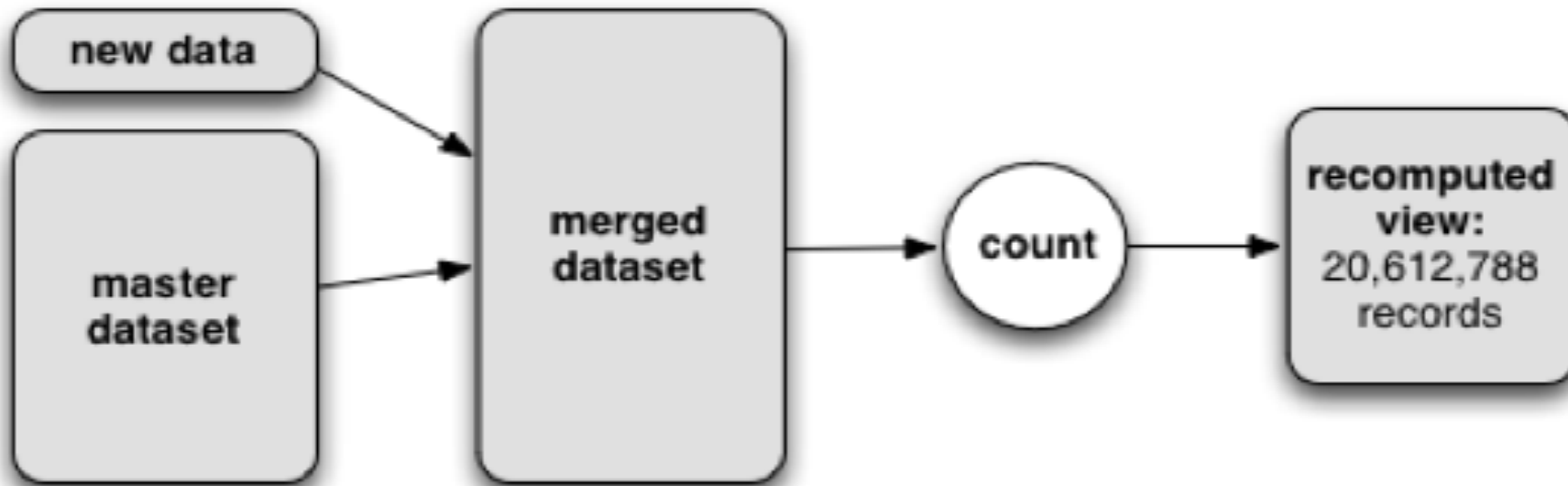
{ exemplo - pageviews por hora }

URL	Hour	# Pageviews
foo.com/blog	2012/12/08 15:00	876
foo.com/blog	2012/12/08 16:00	987
foo.com/blog	2012/12/08 17:00	762
foo.com/blog	2012/12/08 18:00	413
foo.com/blog	2012/12/08 19:00	1098
foo.com/blog	2012/12/08 20:00	657
foo.com/blog	2012/12/08 21:00	101



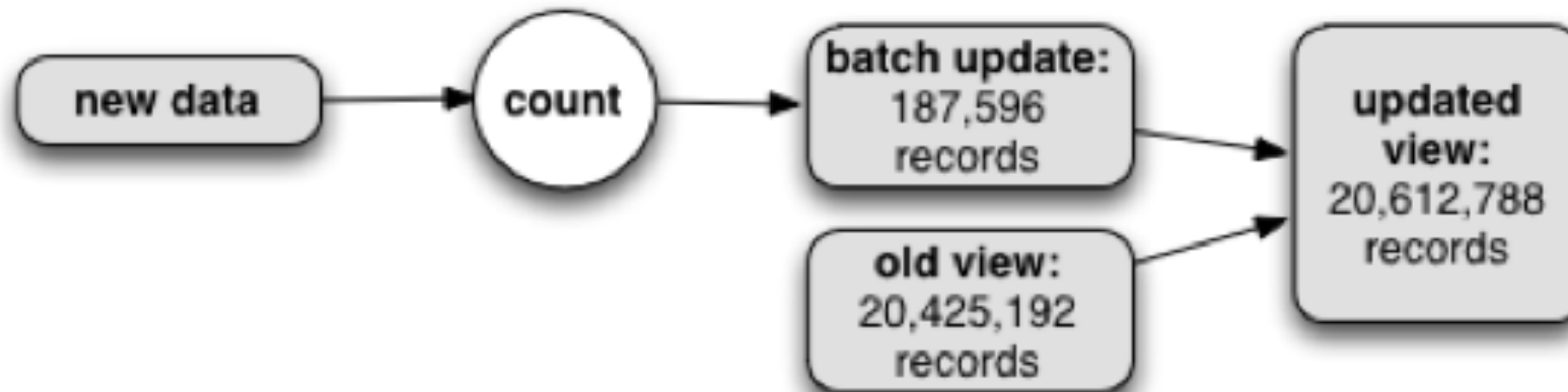
{ algoritmos recomputacionais }

Imaginemos a consulta: “Qual o total de registros do conjunto de dados?”



{ algoritmos incrementais }

Imaginemos a consulta: “Qual o total de registros do conjunto de dados?”



{ algoritmos recomputacionais x incrementais }

- Algoritmos incrementais são sempre a melhor solução?
- É sempre possível utilizar algoritmos incrementais?
- É sempre desejável utilizar algoritmos incrementais?
- Considerações
 - Performance
 - Tolerância a falhas humanas
 - Generalidade do algoritmo

{ algoritmos recomputacionais x incrementais }

"Qual a média de pageviews por URL em um dado momento do tempo?"

URL	# Unique Visitors
foo.com	2217
foo.com/blog	1899
foo.com/about	524
foo.com/careers	413
foo.com/faq	1212
...	...

Recomputation Batch View

URL	# Unique Visitors	Visitor IDs
foo.com	2217	1,4,5,7,10,12,14,....
foo.com/blog	1899	2,3,5,17,22,23,27,...
foo.com/about	524	3,6,7,19,24,42,51,...
foo.com/careers	413	12,17,19,29,40,42,...
foo.com/faq	1212	8,10,21,37,39,46,55,...
...

Incremental Batch View

{ algoritmos recomputacionais x incrementais }

- Sistemas de dados possuem longa vida
- Bugs, **necessariamente**, surgirão e serão instalados em sistemas em produção
- Como o algoritmo de atualização de batch views vai reagir a bugs?
 - Algoritmos recomputacionais são, inerentemente, tolerantes a falhas humanas (basta corrigir e relançar o código)
 - Algoritmos incrementais podem ser seriamente impactados

{ algoritmos recomputacionais x incrementais }

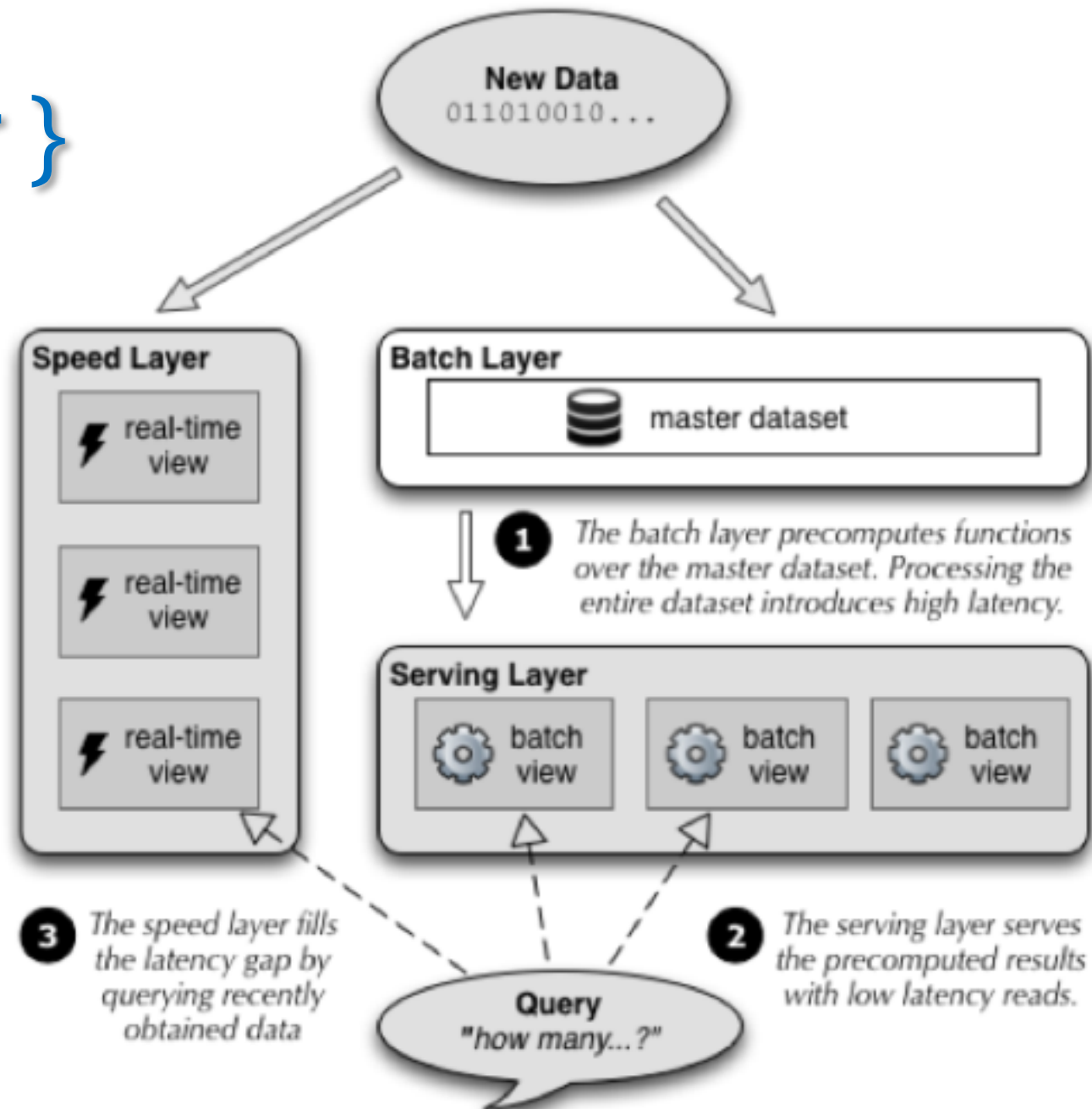
- Algoritmos recomputacionais reconstroem a *batch view* de todo o conjunto de dados
- Portanto
 - Estrutura mais simples da batch view
 - Complexidade da pesquisa “*on the fly*” sobre a batch view é menor
 - Algoritmos recomputacionais são mais genéricos

{ serving/speed
layers }

Serving Layer
Speed Layer



{ serving layer }



{ serving layer }

- Indexar as visões e oferecer interfaces para que os dados pré-computados possam ser **rapidamente consultados**

URL	Bucket	Pageviews
foo.com/blog/1	0	10
foo.com/blog/1	1	21
foo.com/blog/1	2	7
foo.com/blog/1	3	38
foo.com/blog/1	4	29
bar.com/post/a	0	178
bar.com/post/a	1	91
bar.com/post/a	2	568

{ serving layer - requisitos }

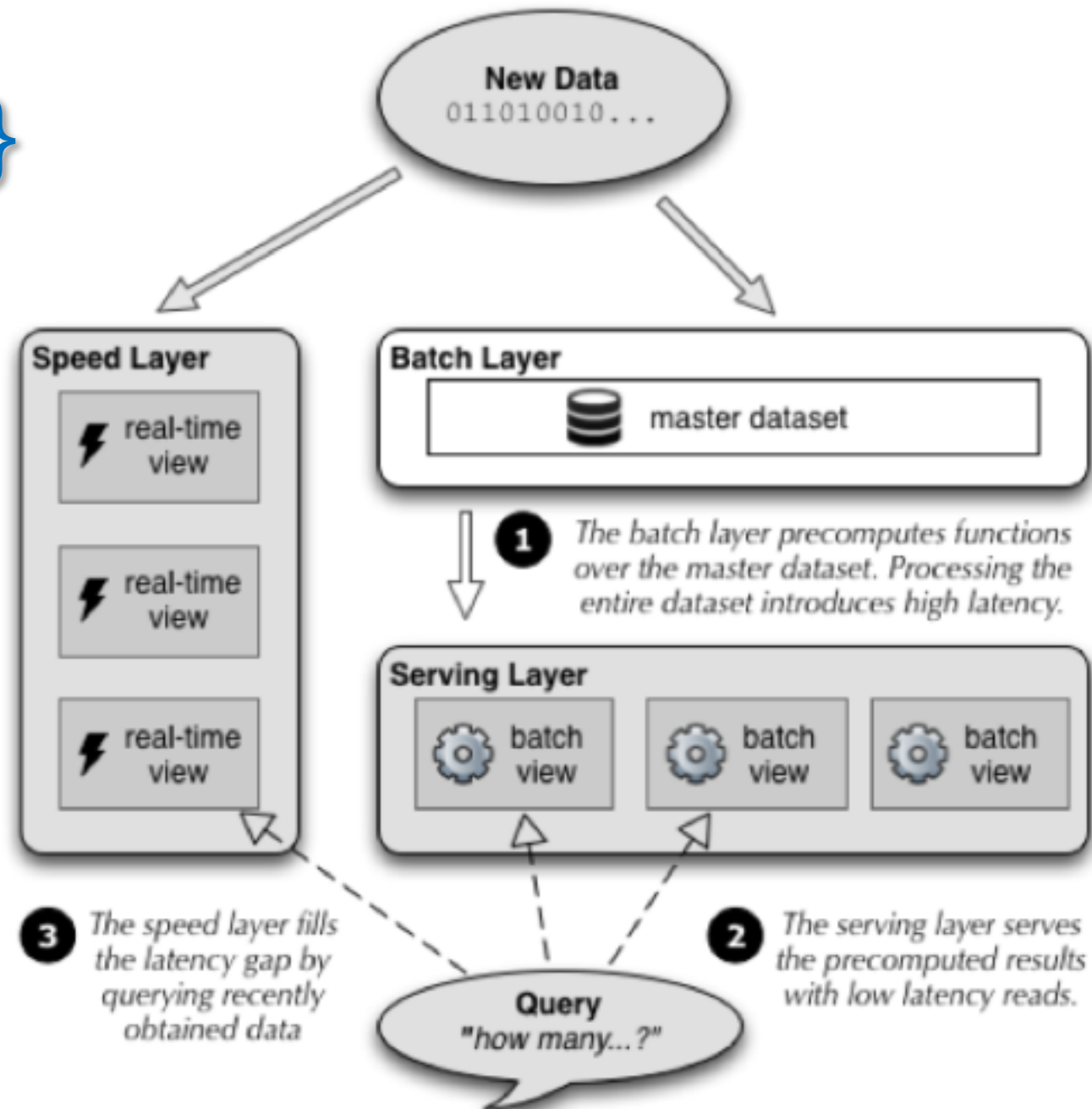
- Escrita em batch
- Escalabilidade
- Leituras randômicas
- Tolerância a Falhas

Por que escritas aleatórias não constam como requisito?

Exemplo: ElephantDB

<https://github.com/nathanmarz/elephantdb>

{ speed layer }



{ speed layer }

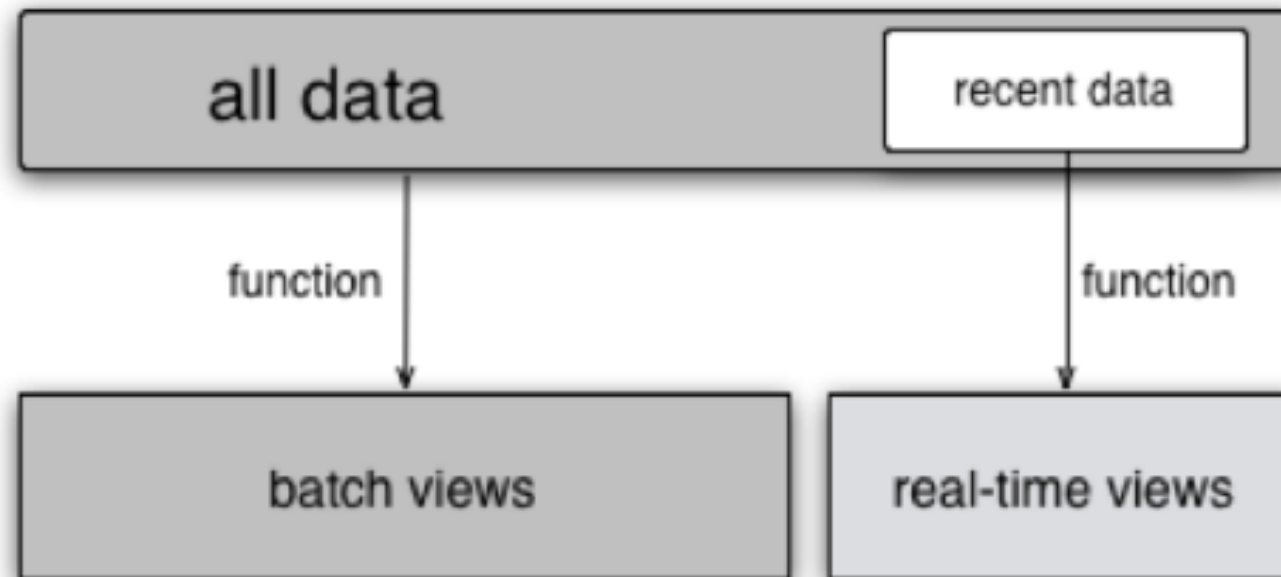
- Oferece consultas de baixa latência sobre dados **recentemente** (e constantemente) atualizados
- Baseado em computação incremental
 - Fator que introduz complexidade significativa, quando comparado com batch layer
- Camada responsável por dados que ainda serão inseridos na Serving Layer
- O conjunto de dados possui, no máximo, poucas horas e é consideravelmente menor que o conjunto mestre de dados

{ speed layer - características }

- Transiente
- Mais complexa, porém, possui maior resistência a erros
 - Informações erradas têm vida curta
- Em soluções relacionais tradicionais "tudo que existe é a speed layer"
- Produz visões que possam ser rapidamente consultadas
 - Visões representam dados recentes e precisam ser atualizadas assim que os dados chegam

{ speed layer }

real time view = function (recent data)



{ speed layer - requisitos }

- Leitura aleatória
- Escrita aleatória
- Escalabilidade
- Tolerância a Falhas

{ speed layer - exatidão eventual }

- Frequentemente é difícil computar funções incrementais
- Vide: "Algoritmos Incrementais"
- Abordagem diferente: aproximar a resposta correta
 - Todos os dados estarão, em breve, representados na Serving Layer
 - Qualquer aproximação é apenas temporária e percentualmente pouco significativa
 - Equilíbrio entre performance e exatidão

{ continua... }

Sempre continua, nunca termina!

Mas em termos de visão geral, ficamos por aqui. ;-)

{ referências }

[1] MARZ, N; WARREN, J. *Big Data – Principles and Best Practices of Scalable Realtime Data Systems*. MEAP Edition, Manning

