GPU TECHNOLOGY CONFERENCE

NVIDIA

ACCELERATED SOFTWARE AS A SERVICE

Mike Houston
Principle Engineer, Mobile and Cloud Computing
NVIDIA

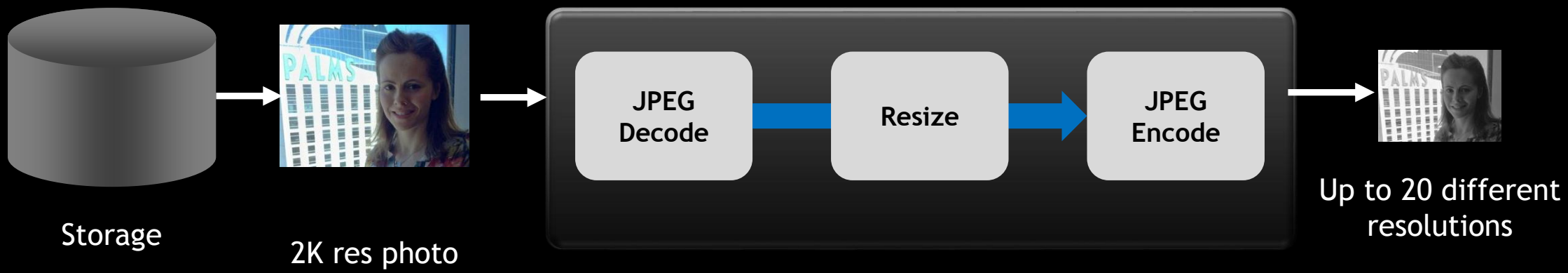
GPU ACCELERATION AS A SERVICE

- Easy datacenter deployment and use
- GPU is a new thing in datacenter
  - Lots of acceleration potential
  - Low level libraries can be challenging to get used and deployed
- Provide acceleration through a simple REST API
  - Equivalent of a dynamic library in datacenter
  - Wrap all functionality and technical magic
  - Easier integration with deployed SW infrastructure

IMAGE RESIZING

Challenge: Heterogeneity at Increasing Scale

Common Ingest Pipeline of today



Facebook + Instagram: 400M Photo Uploads Per Day

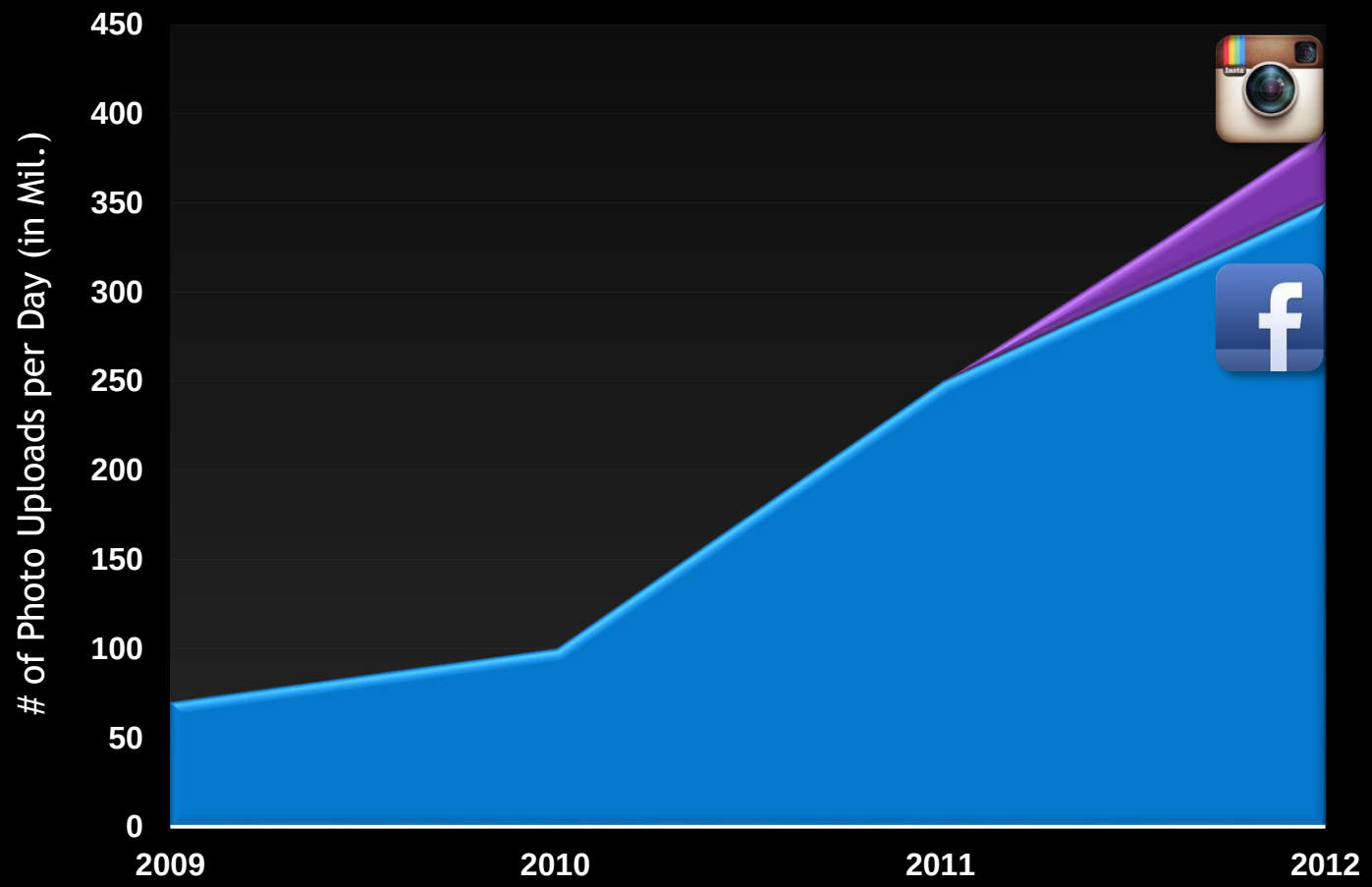
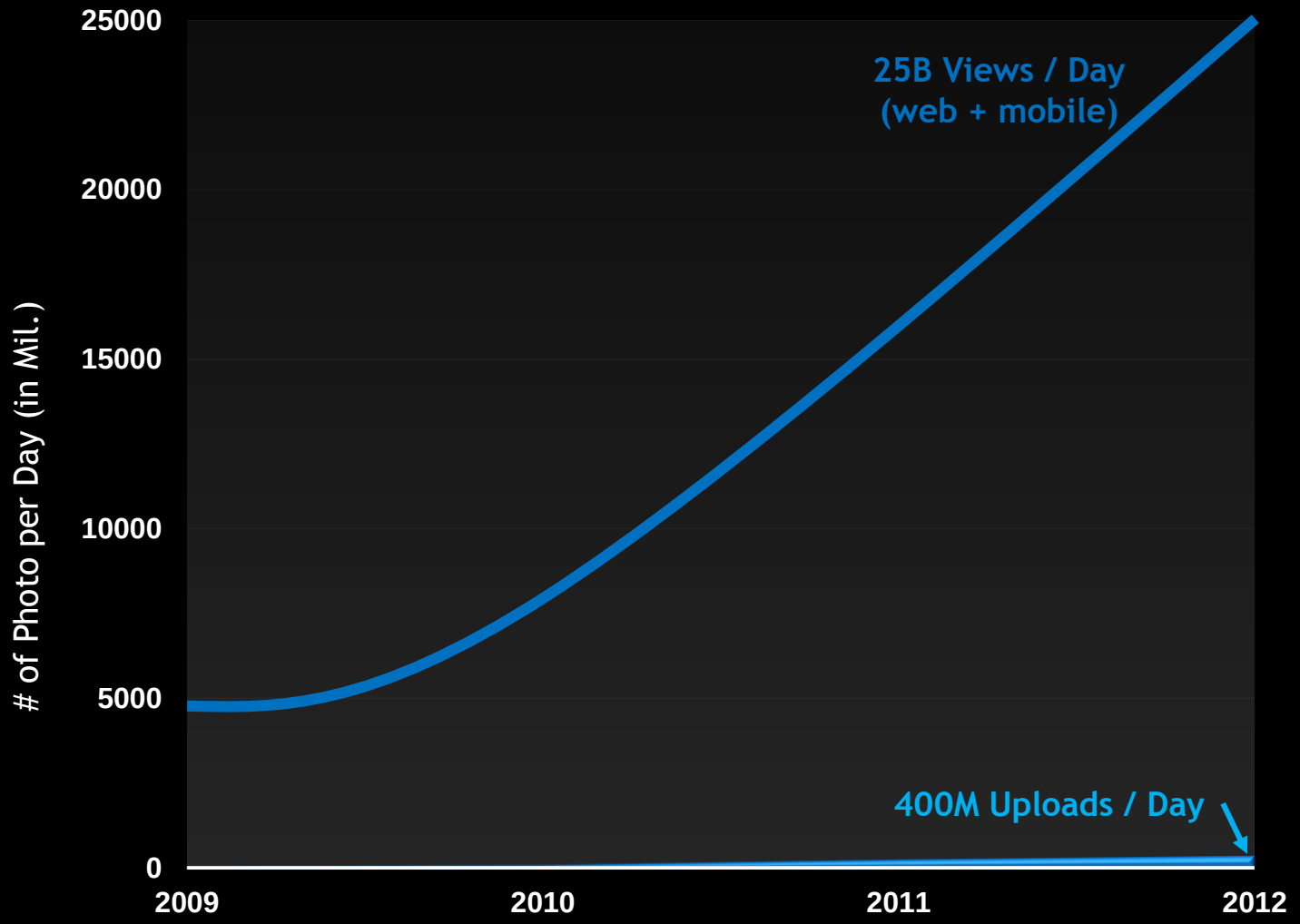


Photo Delivery



WHY FAST RESIZING

- Generate multiple sizes on ingest
  - Reduce server count and handle higher loads
  - Afford to do higher quality filtering
- Generate resizes on the fly
  - Reduce storage costs, especially for the long tail
  - Offer lots of different options for image manipulation
  - Remove need to predefined breakpoints
  - Optimize for a given target device
    - Less bandwidth, faster page load
- Requires high throughput, low latency solution -> GPUs

HOW?

- Key topics we'll cover
  - Request handling
  - Resource access
  - Performance
  - Deployment scenarios
- Punch-line for GPU offload
  - Efficient management and scheduling of resources critical

REST SYNTAX EXAMPLES – HTTP GET

- URL layout:

/<operation>/<width>x<height>/path/to/file.[jpg|jpeg]?param1=value1[;|&]param2=value2...

Examples:

http://sass.com/resize/1024x1024/pix/Growth_of_cubic_bacteria_25x16.jpg

http://sass.com/resize/800x/pix/example.jpg?quality=50&crop=10x10x100x100

http://sass.com/resize/x/pix/example.jpg?quality=75type=progressive&optimize=yes

REQUEST HANDLING 1

- Need something that can handle heavy request load
  - Sometimes referred to as “c10k” problem
  - Leads you towards event based solutions
- Need deep control over resources and scheduling
  - Leads you towards thread based solutions

- Recommendation: use hybrid solution
  - Event model for asynchronous IO
    - High connection concurrency and good failover handling
  - Thread pool for GPU access
    - Allocations, streams, transfers, and kernels belong to a worker thread

REQUEST HANDLING 2

- Scheduling the GPUs critical
  - Want overlapped communication and computation
  - Schedule to multiple GPUs
    - Load balancing
- Split CPU and GPU load
  - Split processing pipeline up to which core makes the most sense
  - CPU
    - JPEG parsing and decode
  - GPU
    - DCT/IDCT, resize, filters, compression, etc.

RESOURCE ACCESS

- Local files
  - Async file i/o
- HTTP backend
  - Talk to backend severs - Amazon S3, webserver
  - Need robust failover handling - more on this later

PLACES TO START - HTTP

- Full solutions to plug into
  - Nginx
  - Apache
  - Varnish
- Roll your own

ROLLING YOUR OWN HTTP SERVER

- Why?
  - No policy imposed on you
  - Better control of GPU resources and scheduling
- How?
  - Boost::ASIO examples
    - "HTTP Server 3"
      - Single IO service and a thread pool
    - "HTTP Client"
      - Asynchronous communication with server
  - HTTP request/response parsing
    - http-parser (Joyent)
  - Nvidia NPP
    - Fundamental image processing routines

CRITICAL OPTIMIZATIONS

- JPEG parser
  - Can quickly become CPU bottleneck when GPU is doing the majority of the work
- Huffman decode
  - Decode can be quite expensive, especially progressive decode
- Huffman tree build – progressive output support
  - Not generally GPU friendly, but offload here helps reduce CPU load
- Latency, latency, latency
  - No batching allowed
    - latency kills
  - Critical scheduling of transfers, kernels, and allocation
    - No blocking CPU or GPU, ever

HOW FAST?

- Compared to GraphicsMagick 1.3.19 + OpenMP + scheduling optimizations
- AWS EC2 g2.2xlarge
  - ~5X the throughput, ~5.5X better latency
- Dual E5-2667 + 2 Tesla K10s
  - ~23X the throughput, ~20X better latency

WEB INFRASTRUCTURE CAN BE TRICKY

- There is the HTTP specification and there is reality
  - You must be super strict when interacting with clients
  - You must be pretty lax when interacting with servers
- Network stack tuning
  - Still needed for heavy connection loads
  - Some distributions have “modes” for this – see RHEL
- Load test!
  - Different tools hit you differently
    - Siege – generate high load and beat on systems
    - Apachebench – easy to use and good statistics
    - Iago – can generate consistent transaction load and excellent statistics
- Security

QUICK INTRODUCTION TO SECURITY

- Errors in your SW you really want to prevent
  - Denial of service attacks
  - Privilege escalation
  - Information leakage
- Server lockdown
  - Do everything you can at the OS level
    - Lots of scripts out there
    - Make sure you are up to date
  - Lock down IPs you talk to
    - If talking only to localhost, lock it down to localhost

QUICK INTRODUCTION TO FUZZING

- What you build may be exposed to the whole world
- Inject lots of errors and really beat on things
- You will likely find
  - Segfaults
  - Infinite loops
  - Races
  - Unexpected behavior

- Makes your code better
- But, makes you bitter

RADAMSA - HTTPS://CODE.GOOGLE.COM/P/OUSPG/WIKI/RADAMSA

- Black box fuzzer
  - Generate lots of negative testing inputs
  - Given exemplar inputs, generate random versions
- Great for testing parsers
  - HTTP
  - JPEG
  - Commands
- Can take a *really* long time to trigger issues
- Use it! (or alternative)

DEPLOYMENT SCENARIOS

- Amazon EC2 – G2.2xlarge instances
- Content Delivery Networks (CDNs)
- Backend infrastructure

AMAZON EC2 – G2 INSTANCES

- Hardware virtualized
- You get ½ a GRID K520 and 8 vCPUs
- Take off the shelf AMI, add CUDA driver/toolkit, launch
  - Ubuntu 12.04 LTS
  - CUDA 6.0
- Integration with current SW infrastructure
  - Run service behind stack on same node
  - Nodes as standalone server

CONTENT DELIVERY NETWORKS

- Massive distributed caching
- Sit behind Varnish/Squid/Nginx as a backend
  - Filter requests
  - Cache output from image server
  - Load balance multiple image servers at the datacenter level
- Talk back to Varnish/Squid/Nginx as server
  - Cache base images
  - Customer's access keys don't go through your SW (S3 buckets)

BACKEND INFRASTRUCTURE

- “Bring your own servers”
  - Add Tesla K10/20/40
- GRID VCA