The continued evolution of High Frequency Trading systems

- Hot technologies in Banking

Richard Croucher
VP High Frequency Engineering Specialised Trading

December 2013

Background and Biases

Platform Architect - experienced at of all layers of the infrastructure and with devices from Smart phones to Mainframes.

Designed and built computers from the chip design upwards – ex. hardware designer

20+ years using and developing UNIX - ex. kernel developer

Programs in multiple languages, having started with Assembler on self built computers.

- Transitioned through Pascal, C, C++, Java, C# and Erlang

Created Cloud management platforms at Sun Microsystems and Microsoft Live

Banking

- Gained experience of Banking IT systems in both consulting and permanent roles with Barclays, Commerzbank, CreditSuisse, DeutscheBank, HSBC, Flowtraders, Merrill Lynch and RBS.
- Specialist on low latency and high frequency trading systems

Technologist not a Banker.

Enjoys discovering and using new technologies

Intro

Trading

Trading systems

Developing Trading systems

- low latency
- C++ expertise
- Java expertise
- RDMA
- GPU’s
- FPGA
- Functional languages

Legal stuff

Barclays does not endorse any vendors or products listed in the following slides

The inclusion of a product name does not imply that the product is in use, or under consideration at Barclays

Investment Banking

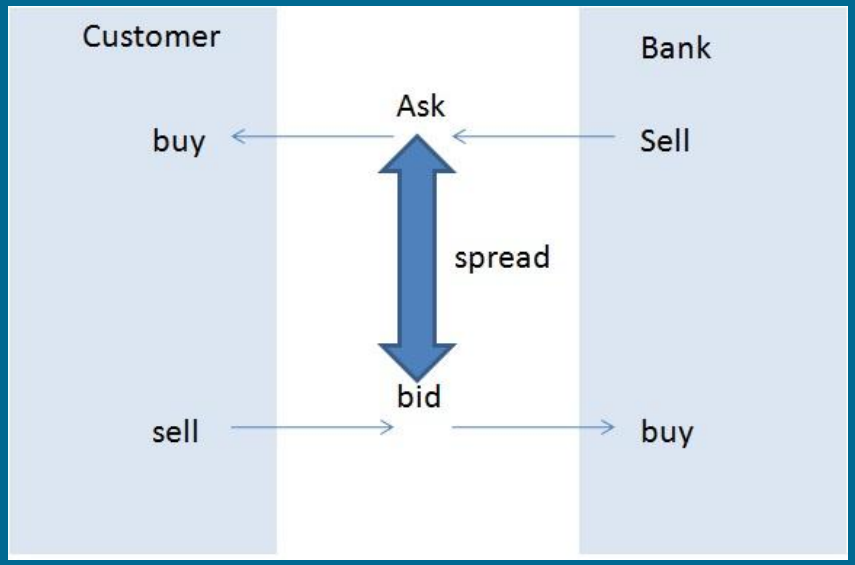
Front Office
•Traders sit on Trading rooms, each with several screens and a fast dialler.
•Punctuated by shouts when someone makes a good deal

Middle Office
•Accountants, Economists, Mathematicians create strategies, watch exposure and risk on client portfolios.
•Trades are matched, cleared and settled

Back Office
•The day's cash movements are aggregated and payment instructions sent out
•The Banks overnight positions are re-calculated and updated

BARCLAYS

It’s all buying and selling



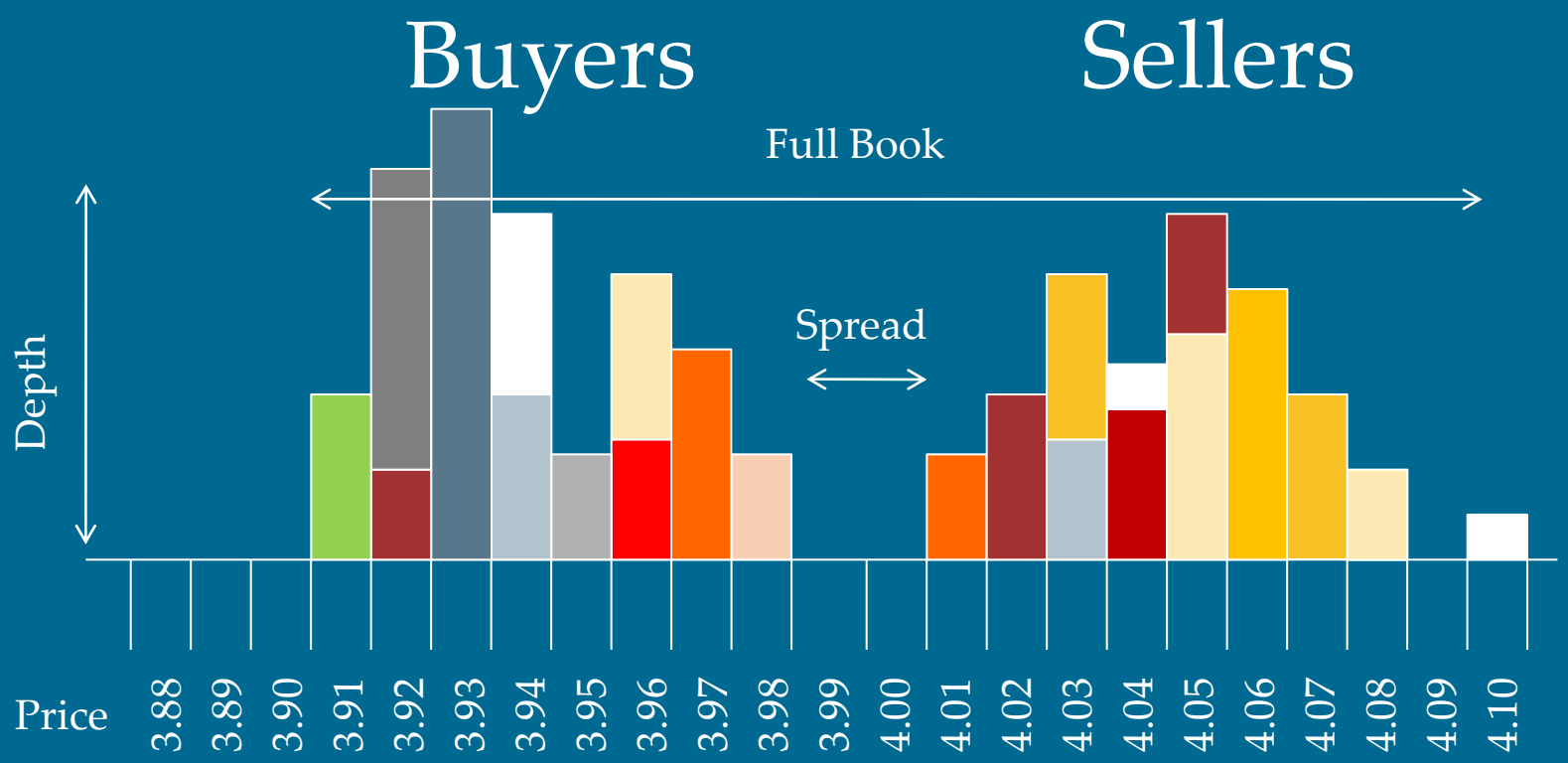
The **venue** is the electronic meeting place where buyers and sellers get connected

The **venue** matches buyers to sellers

The **spread** is the current difference between buy and sell offers

Liquidity increases as buyers and sellers use your venue

Trading book



- Orders can be filled when they match buyers to sellers at the same price
- Market makers improve liquidity by providing offers which sit just outside the current pricing spread
- Lower latency induces shorter spreads and more depth

Trading – main classes

Primary Goal is to make a profit

Alpha is used as a way to express the skill of a trader or the capabilities of an algorithm

- the proportion of the gain made independent of moves of the broader market
- Used as a success indicator or a trading decision

Two primary trading approaches

- Theory based - investment
  - Know your investments
  - Research and news driven
  - Longer term
- Data driven - speculation
  - Statistical and trend driven
  - Can be very short term, even microseconds

Factors impacting liquidy

Liquidity is the most precious resource a venue has

Sellers attract buyers

Buyers attract sellers

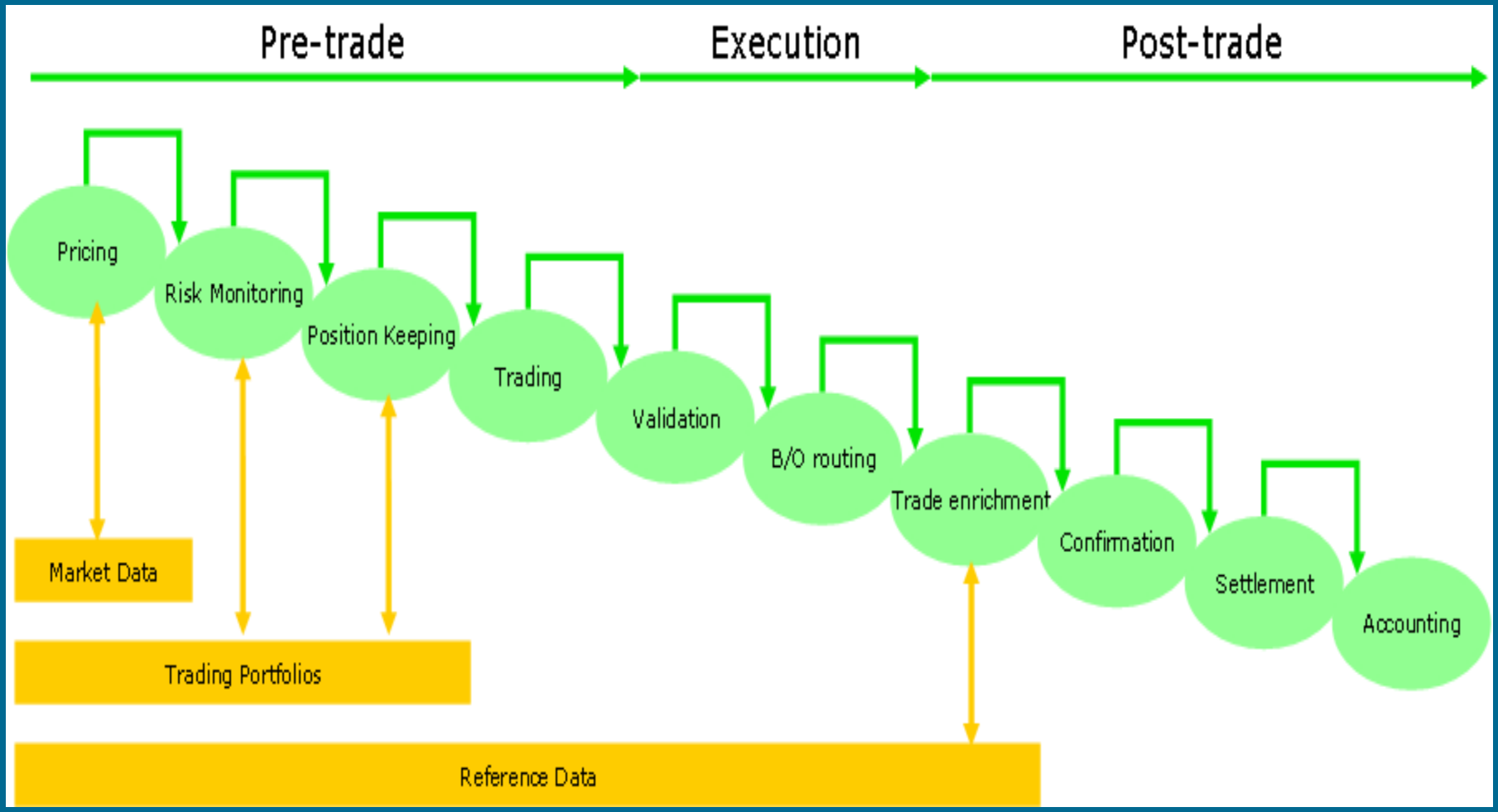
Venues will seek to increase liquidity by:

Offering lower fees

Paying **Market makers** to offer liquidity

Lowering latency to attract algo traders

Trade Flow


Pre-trade
Execution
Post-trade
Pricing
Risk Monitoring
Position Keeping
Trading
Validation
B/O routing
Trade enrichment
Confirmation
Settlement
Accounting
Market Data
Trading Portfolios
Reference Data

Algorithmic Trading Strategies

Arbitration

- Exploiting difference in price for the same stock in different venues

Momentum

- Assumes that a current trend will continue

Alpha (pairs)

- Matches stocks and assumes the value of one should be the same as another
- Based on fundamental macroeconomic statistics

Composites

- Arb a ETF by beating it's update, e.g. A 10% change in BP value on the FTSE100 may translate into a 0.2% change in the overall index

Market Making

- Accepting an Exchange fee to provide liquidity
- Typically large spread just outside current price
- Object is to make money on volume and minimize holding

Buying strategies

Smart Order Routing

- Discover which venue is offering the best price
- A regulatory requirement in EU and USA

Iceberg

- Slice a large order up into lots of small orders to disguise intent and reduce market impact
- Led to a big increase in order volumes and decrease in typical order size

Sweep Order

- Buy only a percentage of desired quantity and pause for more liquidity to be placed, hopefully at the same price

Crossfire

- Liquidity capturing algorithm than targets liquidity within dark pools

Average Prices

Volume Weighted Average Price (VWAP)

- Price * Shares Traded/ Total shares trade
- Primary benchmark for algorithmic trading
- A good buy is when the price is less then the VWAP
- A good sell is when the price is more than the VWAP
- Traders and algorithms are rated on their ability to beat the VWAP

TWAP (Time Weighted Average Price)

- Average price over a specified time
- Used when closing out of a position to make sure they are all sold within a defined time period (a VWAP algo may refuse to sell in a falling market)

Risk Analysis

- **Before Trade**
  - Test strategies with Quant models, often Monte-Carlo simulation on Grids. Backtesting with historical market data.
  - Value at Risk (VAR) estimation, Mark to Market (prices)
  - Implied alpha
  - Long/Short balance
- **During Trade**
  - Client Reporting and performance attribution
  - Transparency
  - Simplicity
  - Absolute returns
- **After Trade**
  - Post trade analytics
  - P&L Stop Loss
  - Sensitivity limits
  - Impact and delay costs

Compliance and Regulations

International Standards and Regulations

- MiFiD, Basel II, Basel III
- Drives standards for :
  - Minimal Capital Requirement for Banks
  - Supervisory Review
  - Market Discipline
  - Operational risk

Regulatory authorities

- FSA – UK Financial Services Authority
- SEC – US Securities and Exchange Commission
- ERG – European Regulators Group (formerly BEREC)

Venue participant control

- “fat finger” protection

Requirement to be able to respond to regulatory investigations e.g. potential insider trading, money laundering or Ponse Schemes

- Drives archive and retrieval requirement
- Heavy fines imposed by failure to respond

Trading Protocols

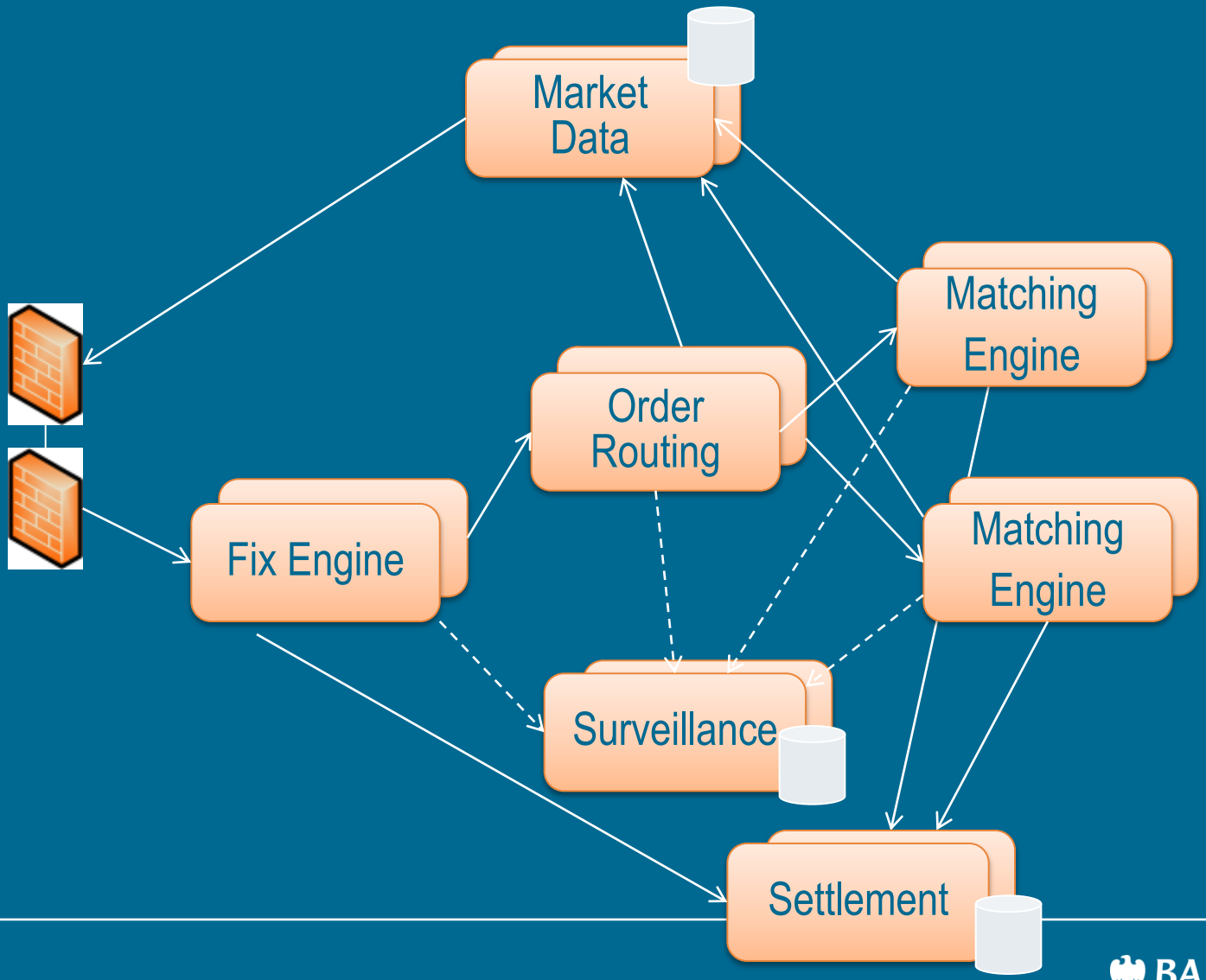
Each venue specifies the format of messages for market data and for orders

- FIX, FAST, ITCH, OUTCH
- Whilst different venues often use the same protocol, each venue has differences in how they are used requiring customized feed handlers for each venue
- Venues frequently make changes to their behaviour, requiring modifications to the feed handlers

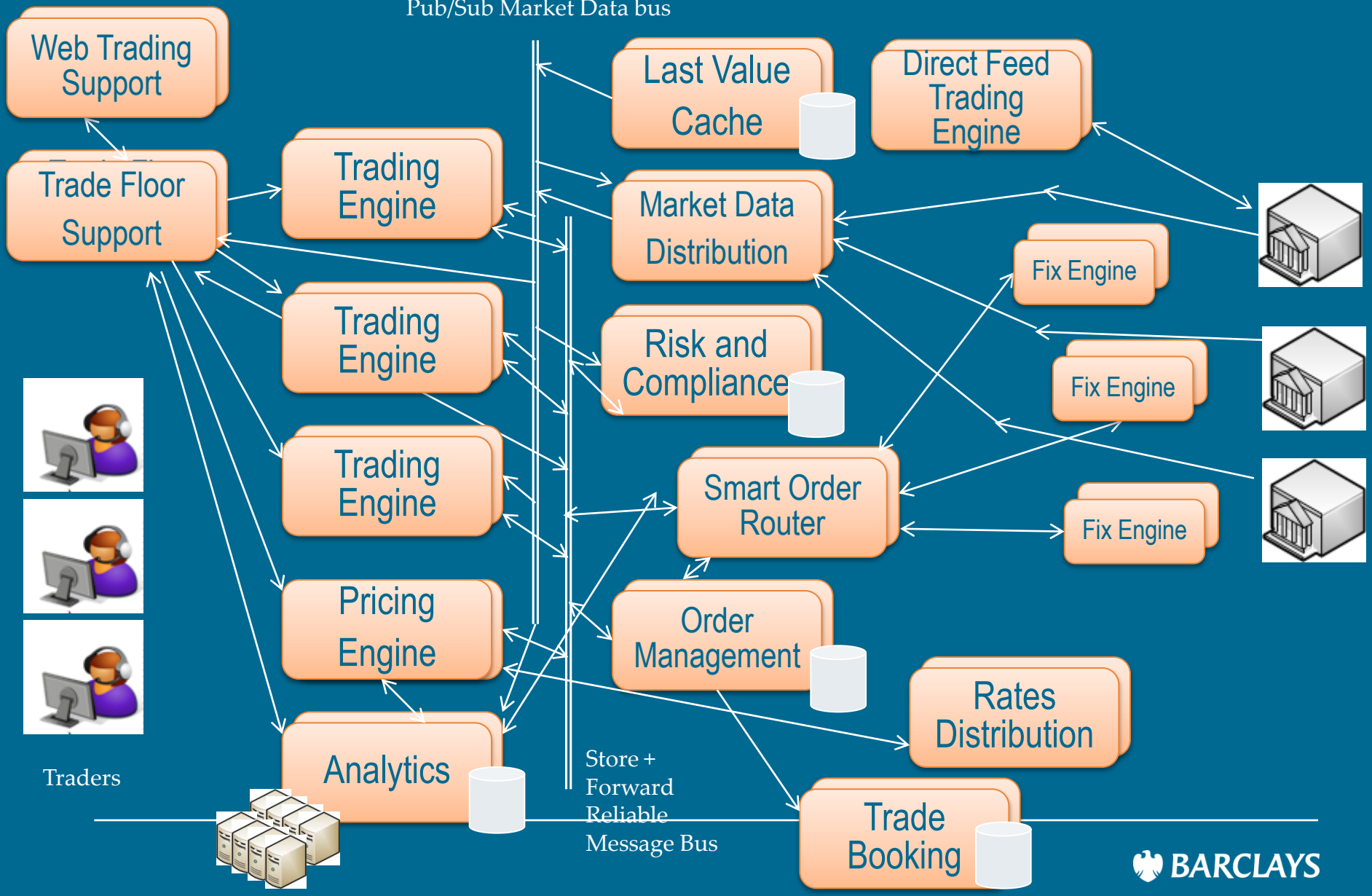
Feed Handlers and FIX engines

- System that interface to the venues
- Products available from Antenna B2bits, Cameron FIX, FIX Flyer, IEISS Fix, Inforeach FIX, Onixs FIX, NYSE Technologies, QuickFIX, Rapid Addition Cheetah,

Sell Side System - Venue



Buy Side System



Time Series – Tick databases

Storing of market data for later analysis and backtesting of strategies

Market data arrives at random intervals, sometimes at huge rates e.g. OPRA has already exceeded 5 million messages/sec

- Data rate too fast for conventional databases
- Data can be sparse so not efficiently stored in conventional databases

Specialised tick database products

  - Kx Systems – KdB
  - OneTick
  - McObject ExtremeDB
- Now moving to noSQL
  - Particularly Cassandra, MongoDB which support higher insert rates

QUANT programming

Now generally used to refer the development of algorithmic trading systems

Still heavy maths bias – Ph.D expected

Expect familiarity and understanding of Black-Scholes model used for derivatives

e.g. to value of a call option for a non-dividend paying underlying stock is

$$C(S,t) = N(d_1)S - N(d_2)Ke^{-r(T-t)}$$

$$d_1 = \frac{1}{\sigma\sqrt{T-t}}\left[\ln\left(\frac{S}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)(T-t)\right]$$

$$d_2 = \frac{1}{\sigma\sqrt{T-t}}\left[\ln\left(\frac{S}{K}\right) + \left(r - \frac{\sigma^2}{2}\right)(T-t)\right]$$

$$= d_1 - \sigma\sqrt{T-t}$$

The price of a corresponding put option based on put-call parity is:

$$P(S,t) = Ke^{-r(T-t)} - S + C(S,t)$$

$$= N(-d_2)Ke^{-r(T-t)} - N(-d_1)S$$

BARCLAYS

Greeks and WAPs

Partial derivatives of the Black-Scholes formulae. Measure the sensitivity of a security with market input variables such as rates, credit spreads, volatilities

- Delta, Gamma, Theta, Vega

Volume Weighted Average Price (VWAP)

- ratio of volume traded to volume trades over a particular period, usually one day. Measure of the average price of a security. Used as a trading benchmark by investors who are attempting to avoid influencing the market with their trades

$$P_{\mathrm{VWAP}} = \frac{\sum_j P_j \cdot Q_j}{\sum_j Q_j}$$

Where Pj is price, Qj is quantity

Time Weighted Average Price (TWAP)

- Average price of a security over a specified time
- VWAP balances volume whilst TWAP balances time

Programming languages and styles

Dominated by C++

Extreme Java programmers also getting good latency results

All Linux based – knowledge of how to tune for low latency and low jitter is important

Interest in functional to improve concurrency

Race to the bottom, e.g. Arbitrage, market making, following strategies are now moving onto FPGA, RDMA

Other strategies moving to deeper research and adopting big data approaches

- Hadoop
- GPU's - Mostly Cuda

Intelligent Trading Architecture example

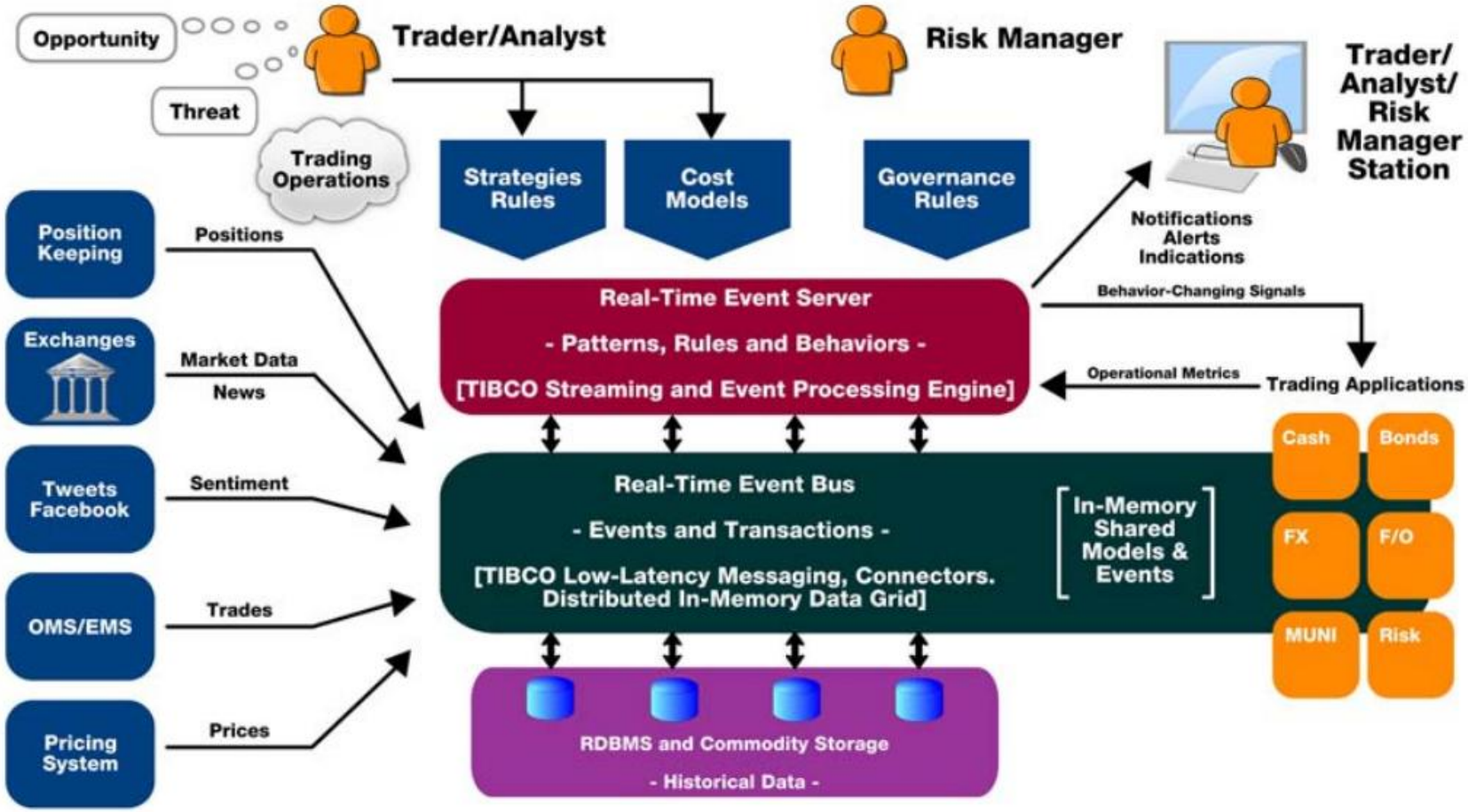


Figure 1: A Reference Architecture for Intelligent Trading

Courtesy of TIBCO

Low latency – programming skills

Packet processing – TCP, UDP, Multicast

Destructor threads – spin waiting for packets to arrive to avoid interrupt wakeup delay

Actor model – minimise lock contention

Nanosecond timestamping

Direct buffer management

Warm up - allocate and preload everything before trading starts

GC impact minimization to avoid jitter

Pinning memory, cache line alignment

CPU isolation and affinity

Achieving durability via replication across network and then persisting

Memory map files when they cannot be avoided

Low latency network programming

Low latency network programming

TCP kernel bypass (BSD socket compatible), usually preload libraries

- Solarflare Onload
- Mellanox VMA, SDP

TCP kernel bypass (proprietary API)

- raw Ethernet socket – handle the headers yourself
- Solarflare ef_vi
- Chelsio WireDirect
- Emulex/Myricom DBL (Data Bypass Layer)

RDMA

- OFED libibverbs, rdma_cm
- OFED rsocket, DAPL, RDS
- Mellanox VMA

Java expertise area

Low latency tuning and jitter avoidance

Expertise with NIO, particularly with buffers

Lock detection and tuning

Network processing - TCP, UDP, Multicast

Packet processing - raw, pcaps

RDMA - Java FastSockets

GC tuning

Tuning tools

- jRocket mission Control
- Intel Vtune
- NeBeans profiler
- SeaNet

C++ expertise area's

Boost – particularly Math

C and even assembler inline

QuantLib - Greeks library

Lockfree++, actor patterns

Performance tuning with pragma's

Compiler optimization

Network processing – TCP, UDP, Multicast

Kernel bypass - Onload, ef_vi

TCP bypass - RDMA

Memory optimization and tuning – cache alignment, large pages

Remote Direct Memory Access (RDMA)

Hardware enabled reliable data transfer across the network

Overcomes the TCP/IP bottleneck and achieves wire speed and lowest latency - 56Gb/s (measured 6Gbyte/s, 2µS latency)

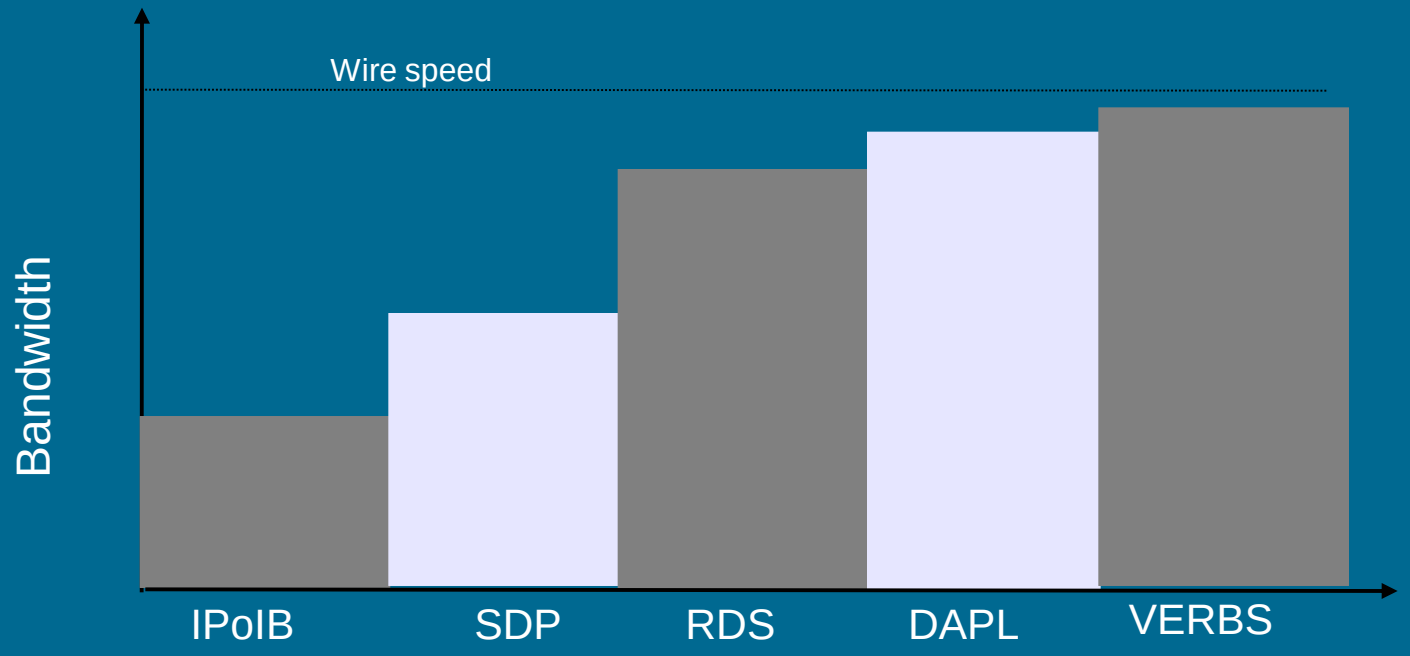
Initially just InfiniBand but now RDMA over Ethernet with RoCE and iWARP

Pre-load SDP or VMA to bypass TCP/IP stack for TCP sessions

For optimal performance need to change API's to use zcopy and RDMA transfer direct between Application memory spaces. Several API's of varying degrees of complexity and performance provide this

- Reliable Datagram Sockets (RDS) – Used by Oracle for RAC
- Direct Access Programming Library (DAPL) – used by IBM for DB2
- Libibverbs – fastest access, near native access to HW VERBS – used by Websphere LLM, TIBCO FTL, Millenium IT, IBM GFS
  - Able to support both ping-pong and pub-sub Use cases

RDMA performance Results by API or Upper Layer Protocol

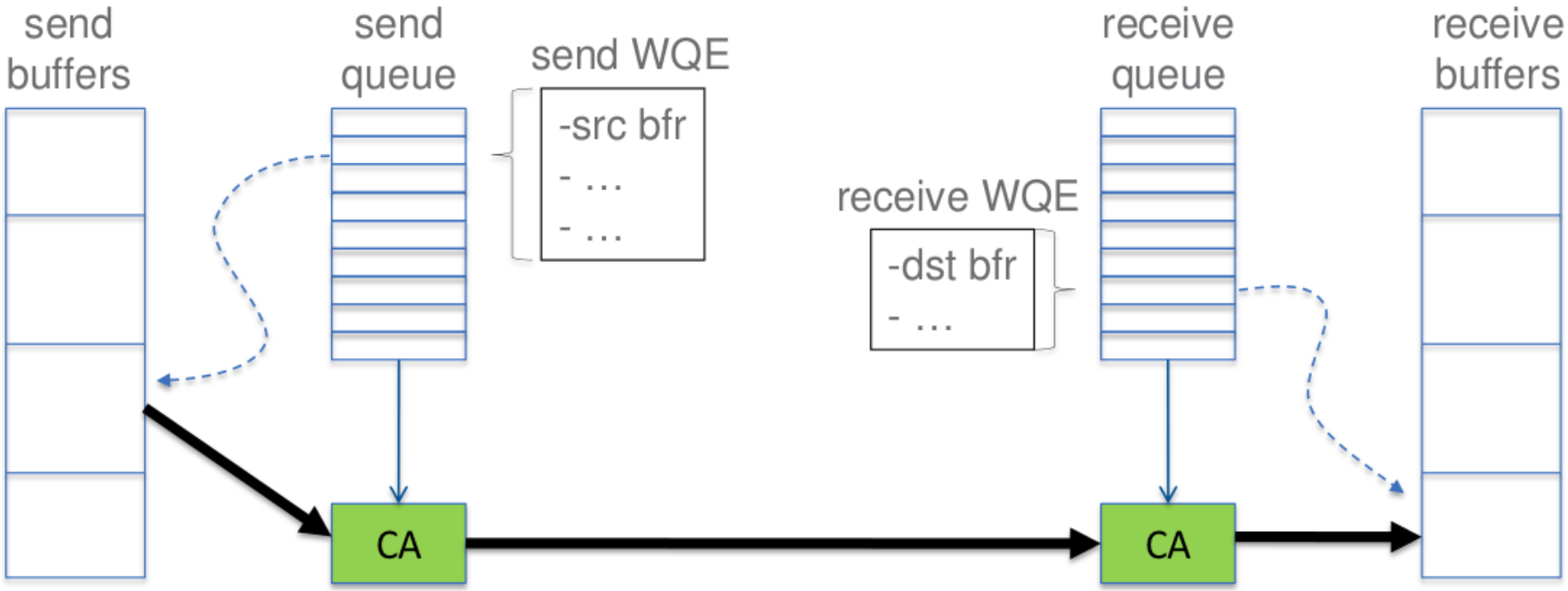


Programming with RDMA VERBS

Four phases to a RDMA program

- Connection management and establishment
  - rdma_cm allows TCP/IP address space to be used to establish ‘queue pairs’ which are used as end points.
- Memory registration
  - locks the memory region which will send or receive data into memory to prevent page faults. HCA loads TLB’s to translate between virtual to physical address
  - exchange memory keys to permit remote access to this memory by the key holder
- Data transfer
  - queue a work request , e.g. Read from or write to a remote server
    - Single block or scatter gather
  - actual data transfer carried out by hardware at wire speed
    - Up to 2GB in a single transfer, HW error detection and correction prevents errors
    - measured at 6Gbytes per second, < 2µS latency across network for 2KB transfers
- Completion
  - Receive or check for notification of completion

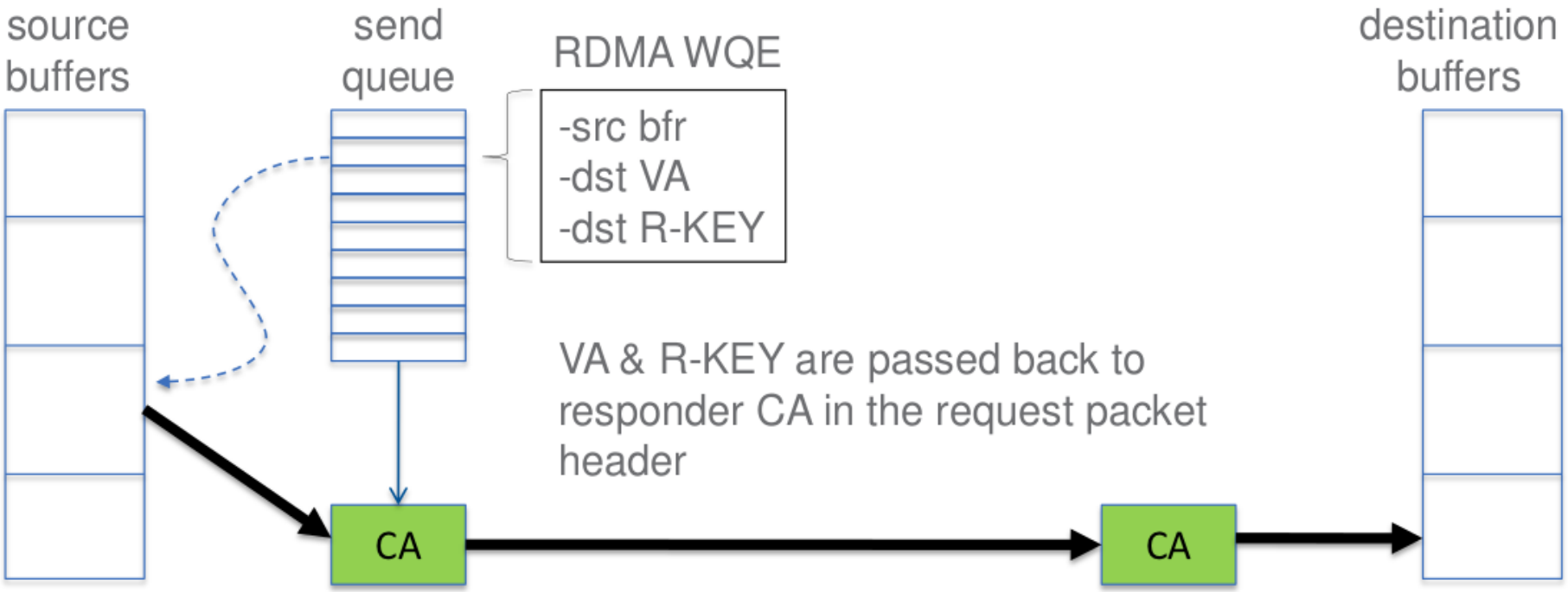
SEND/RECEIVE



-Send WQE defines source buffer

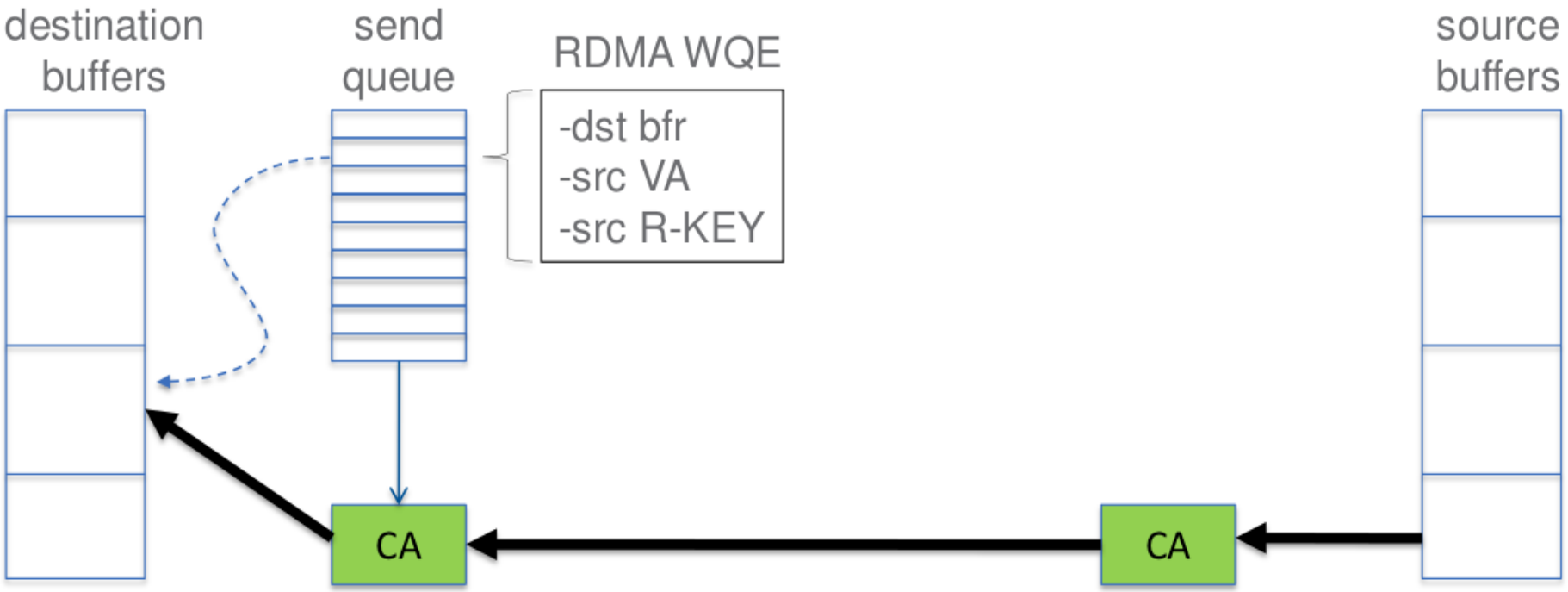
-Receive WQE defines destination buffer
-Receive buffers must be pre-posted

RDMA WRITE



- RDMA WQE defines source buffer and destination buffer addresses
- Responder CA resolves the destination buffer VA into a physical address

RDMA READ



-RDMA READ is exactly the same, except for data transfer direction

GPU

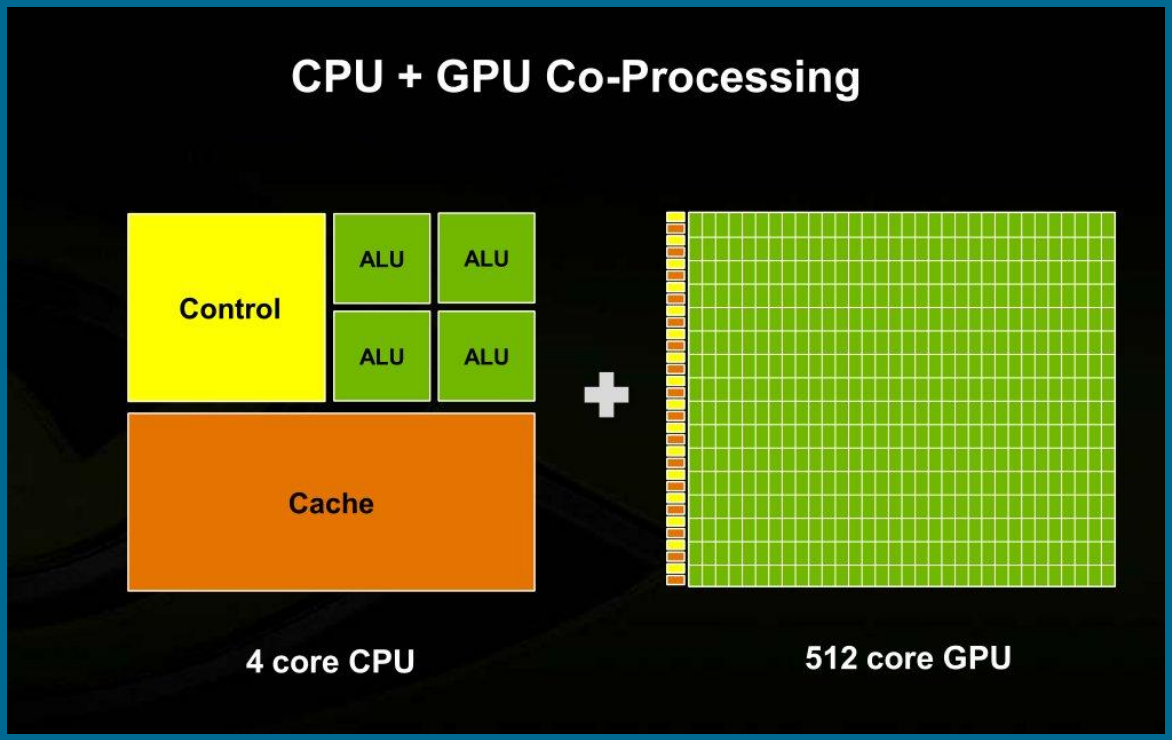
Utilizing Graphics processors for algorithmic compute

OpenCL supports Nvidia, Intel and AMD GPU's

Leader nVidia CUDA (Compute Unified Device Architecture

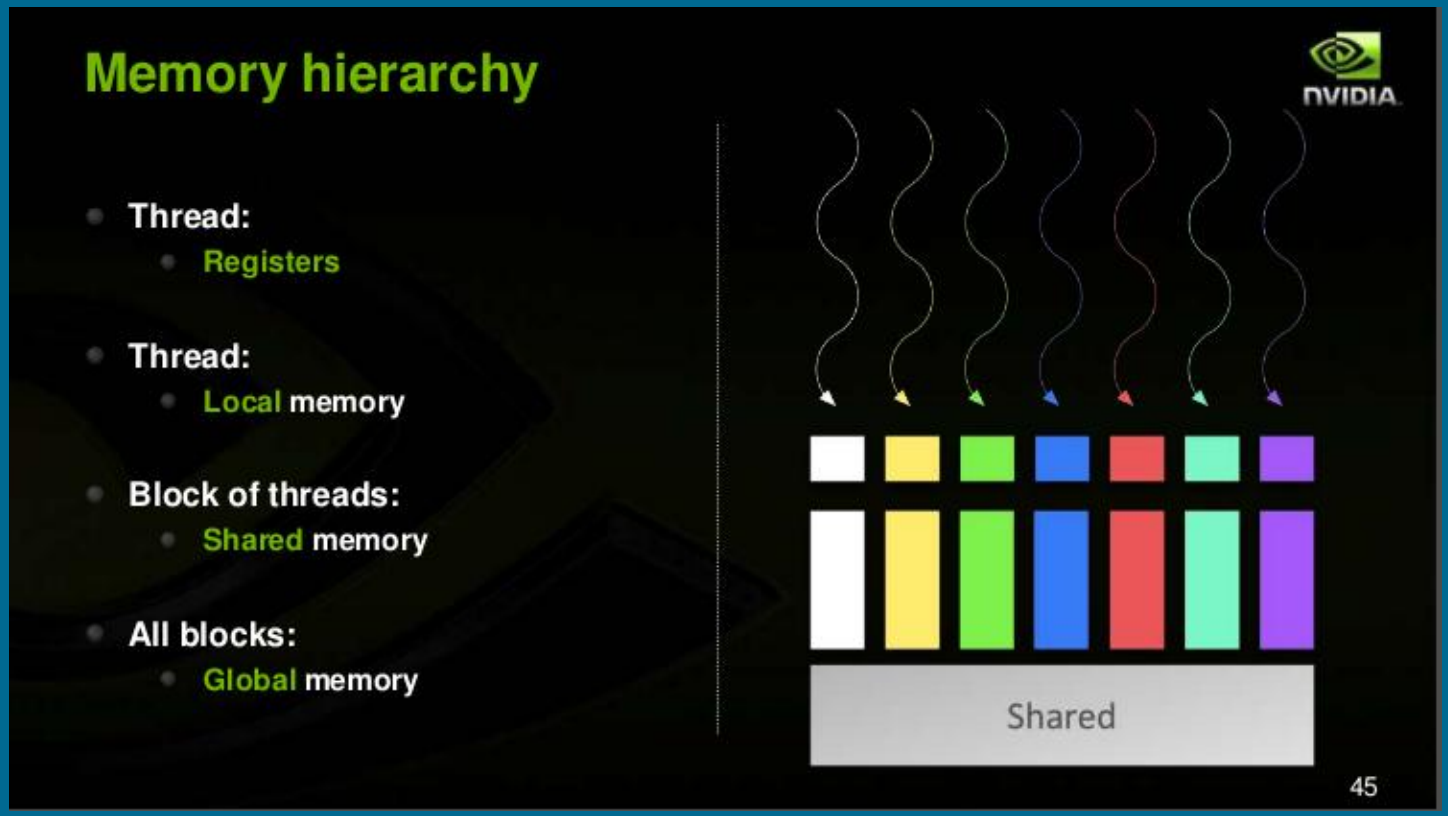
- C/C++ API
- Parallel Nsight (Visual Studio), Visual profiler and cuda-gdb for Linux

GPU



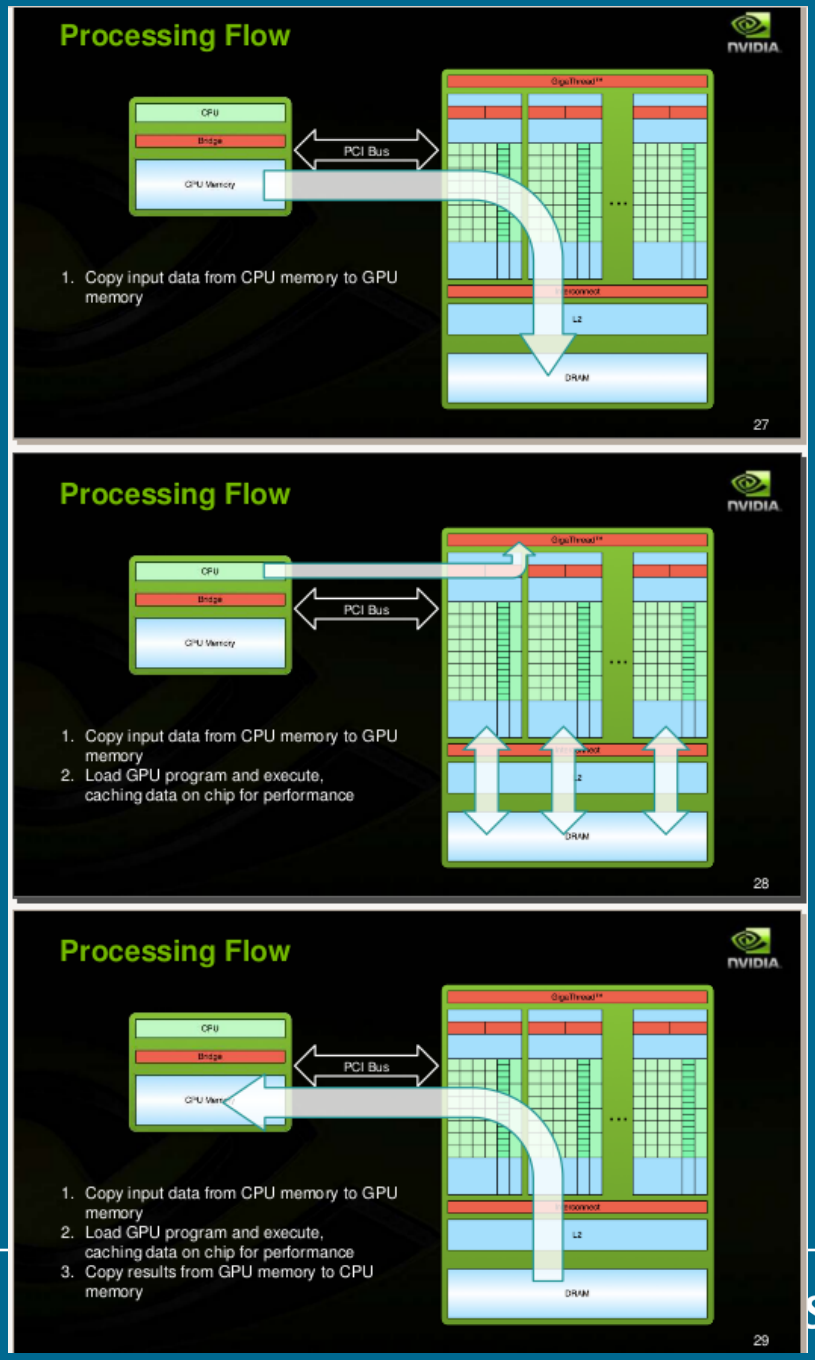
GPU is a SIMD (Single Instruction Multiple data) architecture
Supports large number of parallel threads but EACH thread executions the same code using a different slice of data

GPU memory


Memory hierarchy
NVIDIA
Thread:
Registers
Thread:
Local memory
Block of threads:
Shared memory
All blocks:
Global memory
Shared
45

GPU flow

1. Copy input data from CPU memory to GPU memory
2. Load program code onto GPU. Note: same code executes on each thread
3. Start program execution and wait until finished
4. Copy results back from GPU and merge results



Xcelerit

Programming approach which creates common code for execution on traditional cores and/or GPUs -

Data flow style using Actors

Examples

- Monte-Carlo European Option Price shows speedups of 80-210x when 2x Nvidia Tesla GPU's added
- LIBOR Swap option portfolio, evolution of 80 forward rates across 40 steps using Monte Carlo paths. Used with Greeks to show its sensitivity to underlying interest rate (λ) showed speedups of 60-820x using 2 Nvida GPUs

See Monte-Carlo Methods in Finance using Xcelerit Platform

Xcelerit Code example

```
double endCallValue(OptionData o, double samp); // computes the call-option value at expiry
void generateSamples(double* samp, ...); // generate samples
void writeResults(string file, ...); // write results to file
int main() {
...
generateSamples(&samples , numOpt , numPaths);
for (int o = 0; o < numOpt; o++) {
        optVal[o] = 0.0;
         for (int p = 0; p < numPaths; p++) {
                  // path calculation
                OptionData opt = options[o];
                double callVal =
                        endCallValue(opt, samples[o*numPaths+p]);
                 double val = exp(-R*opt.T)*callVal;
                // reduction
                optVal[0] += val / numPaths;
        }
}
writeResults("output.txt", optVal , numOpt);
```


BARCLAYS

FPGA

Fuse Programmable Gate Away

- An array of electronic gates that can be configured under software control
  - Requires deep hardware knowledge of digital electronics

Two primary vendors of high performance FPGA's with broadly equivalent parts

- Xilinx (Virtex-7), Altera (Stratix V)

Many hardware vendors who manufacture PCIe cards based on these with varying levels of support and capabilities

Pre-designed IP modules cover increasingly sophisticated capabilities

- PCIe bus management, , soft cores e.g ARM and PowerPC, Network ports, TCP/IP stacks, FIX protocol support

Gate speed is slower than ASIC, e.g. Typically 500Mhz clock speed

- Need to parallelize using data flow style approach to compensate

Advantages packet processing since it can start as the packet arrives. Many systems running simple algorithms in under 1µS

FPGA – Hello world (actually blinking light)

VHDL Example

- Compile
- Assign – pin allocation
- Configuration – downloads code onto hardware
- Start – run the code

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity hello_world is

port (
    clk     : in std_logic;
    led     : out std_logic
);
end hello_world;

architecture rtl of hello_world is

    constant CLK_FREQ : integer := 20000000;
    constant BLINK_FREQ : integer := 1;
    constant CNT_MAX : integer := CLK_FREQ/BLINK_FREQ/2-1;

    signal cnt      : unsigned(24 downto 0);
    signal blink    : std_logic;

begin

    process(clk)
    begin

        if rising_edge(clk) then
            if cnt=CNT_MAX then
                cnt <= (others => '0');
                blink <= not blink;
            else
                cnt <= cnt + 1;
            end if;
        end if;

    end process;

    led <= blink;

end rtl;
```

See http://www.jopdesign.com/cyclone/hello_world.pdf

FPGA Programming

Mix of hardware and software skills

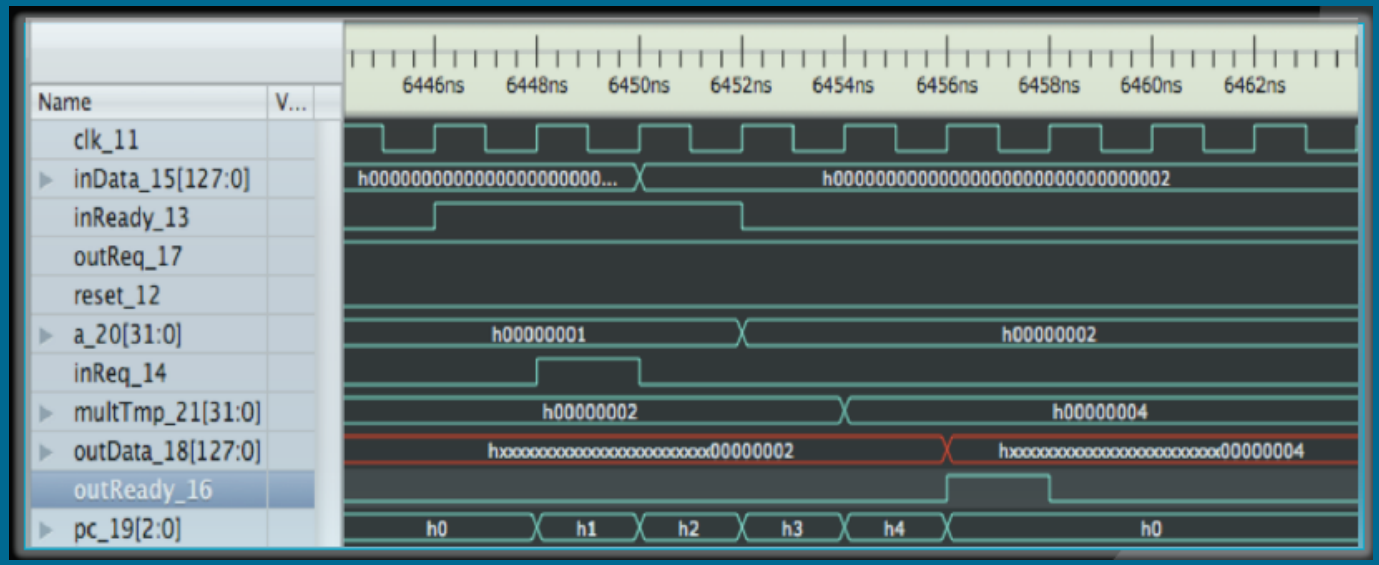
1. Hardware selection – device, board
2. Design - VHDL
3. Simulation
4. Synthesis
5. Test bed instantiation
6. Pin assignment
7. FPGA bitstream generation and program load
8. Test

See Sven Anderssons "How to design an FPGA from scratch" published in EE Times, good place to start although dated

FPGA Programming

Mix of hardware and software skills

1. Hardware selection – device, board
2. Design - VHDL
3. Simulation
4.
5.
6.
7.
8.



See Sven Anderssons "How to design an FPGA from scratch" published in EE Times, good place to start although dated

FPGA - buy or build?

Buy an existing FPGA based product

- Feed Handler appliances– Exegy, NovaSpark, Fixnetix, Activ Financial
- Messaging appliances – Solace, Tervela, TIBCO

Customizable kits for Trading– Algo-Logic, Celoxica, Insight Design Labs, Stone Ridge Technology

High level development kit – Maxeler

IP rich boards – PLDA, Dini Group, Bittware, Intilop

FPGA – programming tools

High level languages

- Bluespec Semu – development environment , HL language with C/C++ integration
- Maxeller MPT – development environment, language and run time appliance
- IBM Liquid Metal - single unified, java like, programming language for CPU, GPU and FPGA

C/C++ integration

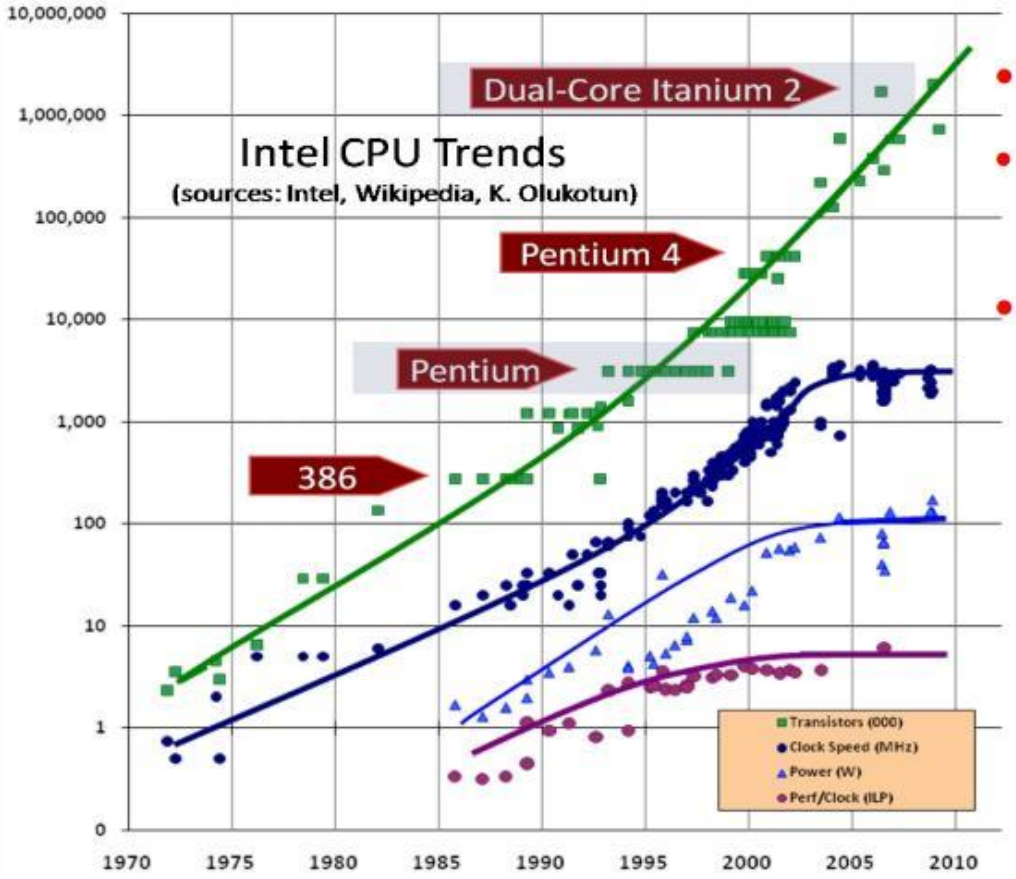
- C-Verilog – Open source library
- Impulse - C/C++ to VHDL, 'R' to VHDL

Algorithms in MaxCompilerMPT

- Build in-hardware algorithms using MaxCompilerMPT
  - Dataflow technology allows for simple programming
  - Easily integrate with high-performance exchange interfaces
  - Programmers can concentrate on business logic
- HFT "top of order book" execution in MaxCompilerMPT:


```
public TradeDecision getTradeDecision(OrderBook order_book) {
    OrderBookLevel bid = order_book.bid[0];
    OrderBookLevel ask = order_book.ask[0];

    HWVar secId = io.scalarInput("securityId", hwInt(32));
    HWVar highPrice = io.scalarInput("highPrice", hwUInt(32));
    HWVar lowPrice = io.scalarInput("lowPrice", hwUInt(32));

    TradeDecision d = new TradeDecision();

    d.Buy = ask.Price <= lowPrice & order_book.securityId === secId;
    d.Sell = bid.Price >= highPrice & order_book.securityId === secId;
    d.Quantity = d.Buy ? ask.Quantity : bid.Quantity;
    d.Price = d.Buy ? ask.Price : bid.Price;

    return d;
}
```

The Future is Parallel – Get Used To It



- Free-Lunch-Is-Over graph*
- Trends in Moore's Law, CPU speed, power, and ILP
- These trends are pushing us into a parallel world
  - Whether we like it or not

*Graph courtesy of Herb Sutter

Functional Languages

Scalability, particularly concurrency is too hard with imperative programming languages

- Servers supporting 1000 hardware threads are already available.
- Trend will increase with Moore's law doubling this every 18 months

Functional Languages are a better match to create scalable code to run on multiple cores

Functional Languages

- Lisp – the granddaddy (c1958) but overtaken now by newcomers
- Haskell is (arguably) the purest functional language, uses threads and Mvars for concurrency
- Clojures - Lisp-1 style, which leverages existing JVMs and Windows CLR. Uses identities which have immutable state for concurrency
- Scala – object functional running on JVM and Android, relies on Actors for concurrency.
- Erlang has strongest concurrency model, designed by default to avoid side effects – variables are immutable. Integrated cluster support simplifies remote execution
- F# - preserves .NET investment but mixes functional, imperative and object programming styles, relies on Async.Parallel to provide explicit parallelism
- Groovy – scripting language leveraging Java platform and accepting Java code
- Java – Functional JAVA fork available now. Lambda expressions due in Java 8

Erlang concurrency example

```
-module(tut14).
-export ([start/0, say_something/2]).

say_something(What,  Times) ->
        io:format("~p, ", [What]).
start() ->
        spawn(tut14, say_something,  [Hello, 3]),
        spawn(tut14, say_something, [goodbye, 3]).
```

Now run it

- `tut14:say_something(Hello, 3).`

Hello, Hello, goodbye, Hello, goodbye, goodbye,

Erlang distributed computing – ping/pong

```
ping(0, Pong_PID) ->
        Pong_PID ! finished,
        io:format("ping finished~n", []);
ping(N, Pong_PID) ->
        Pong_PID ! {ping, self()}
        receive
                pong ->
                        io:format("Ping received pong~n", [])
        end,
        ping(N -1, Pong_PID).

pong() ->
        receive
                finished ->
                        io:format("Pong finished~n", []);
                {ping,  Ping_PID} ->
                        io:format ("Pong received ping~n", []),
                        Ping_PID ! Pong,
                        pong()
        end.

start() ->
        Pong_PID = spawn(tut15, pong, []),
        spawn(tut15, ping, [3, Pong_PID]).
```

Functional Languages - Erlang

Benefits of Erlang

- Erlang OTP provides comprehensive runtime support including Debugger, Event Managers, Watchdogs, FSM, in-memory DB, Distributed DB, HA, Unit test, Docs, Live Update
- Big **Int** – no need to deal with overflows
- Built in support for HTML and SNMP
- Powerful bit level processing
- Code is more powerful – achieves more in fewer lines
- Preference for automated restart – fail early strategy significantly reduces explicit try/catch coding
- Vast ecosystem of Erlang code

Challenges with Erlang

- Immutable variables take some getting used to
- Forces you to use recursion since no 'while', 'for' operations
- Native String handling inefficient - text intensive systems use bit strings
- Overhead of typed data reduces efficiency e.g. Int's
- Virtual Machine, GC, Interpretation overheads although HiPE provides optimized support for Unix on x86.

Practical experience is that inherent concurrency more than compensates for VM overhead for most real work. Exceptions are intensive numeric calculation and ultra low latency trading

- Use Erlang as a Control Plane to achieve concurrency then drop down into C/C++ code where performance is critical
  - Code as Ports, or Native Functions; allows these to be single threaded but Native Functions must run quickly in order not to disrupt Erlang scheduling

Mobile

Whilst not part of Trading it's an area of rapid growth in the Banks

Mobile payments - custom apps to handle your Bank account and to make payments - Pingit

Long term trend to decrease dependence on high street branches

BYOD – Bring your own Device

- Banks encouraging employees to use their personal devices rather than company issue Blackberries
- Apps developed to allow increasing access to Bank data in a safe and secure manner

Questions?

Feedback

Richard.croucher@Barclays.com