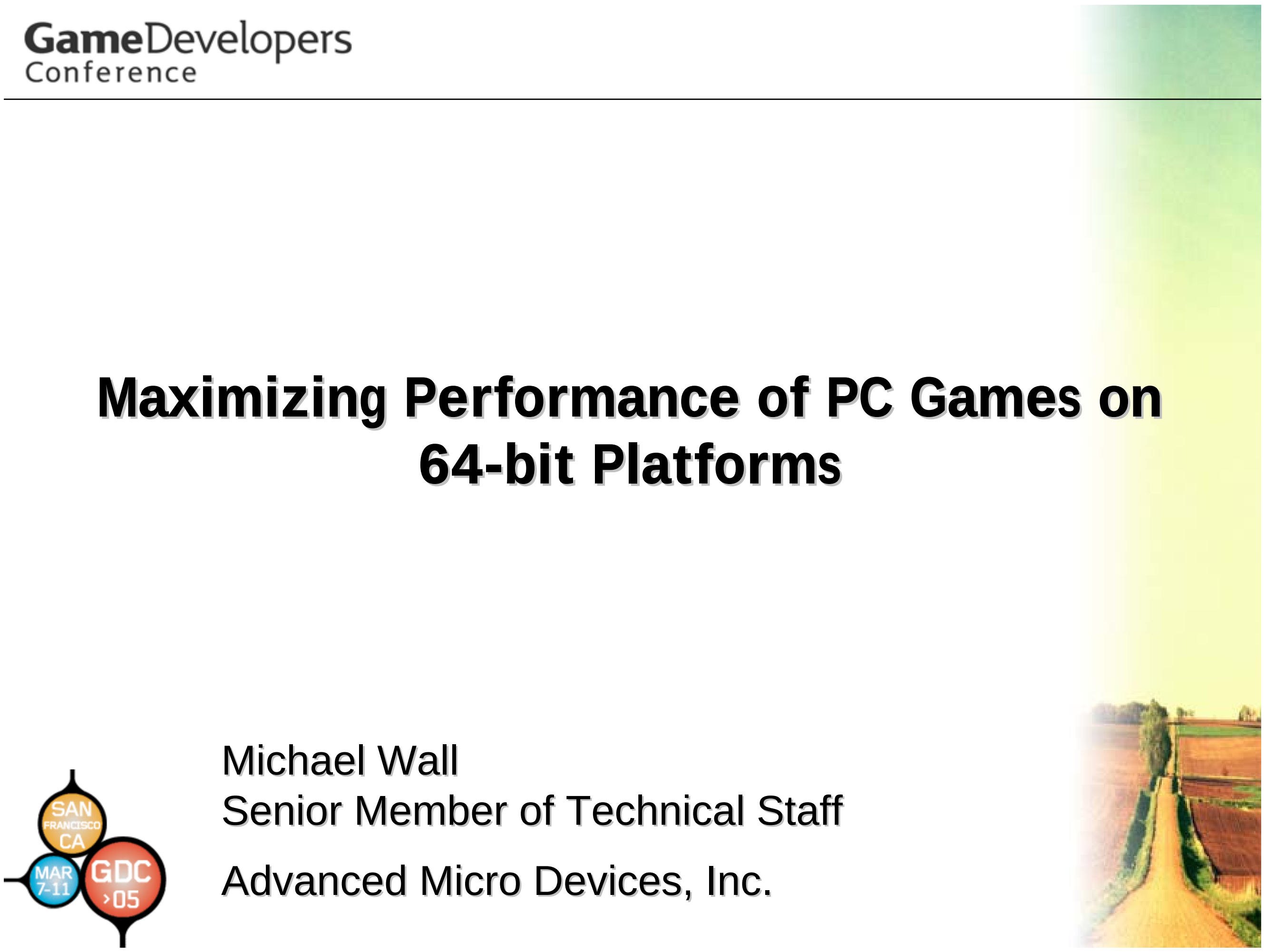
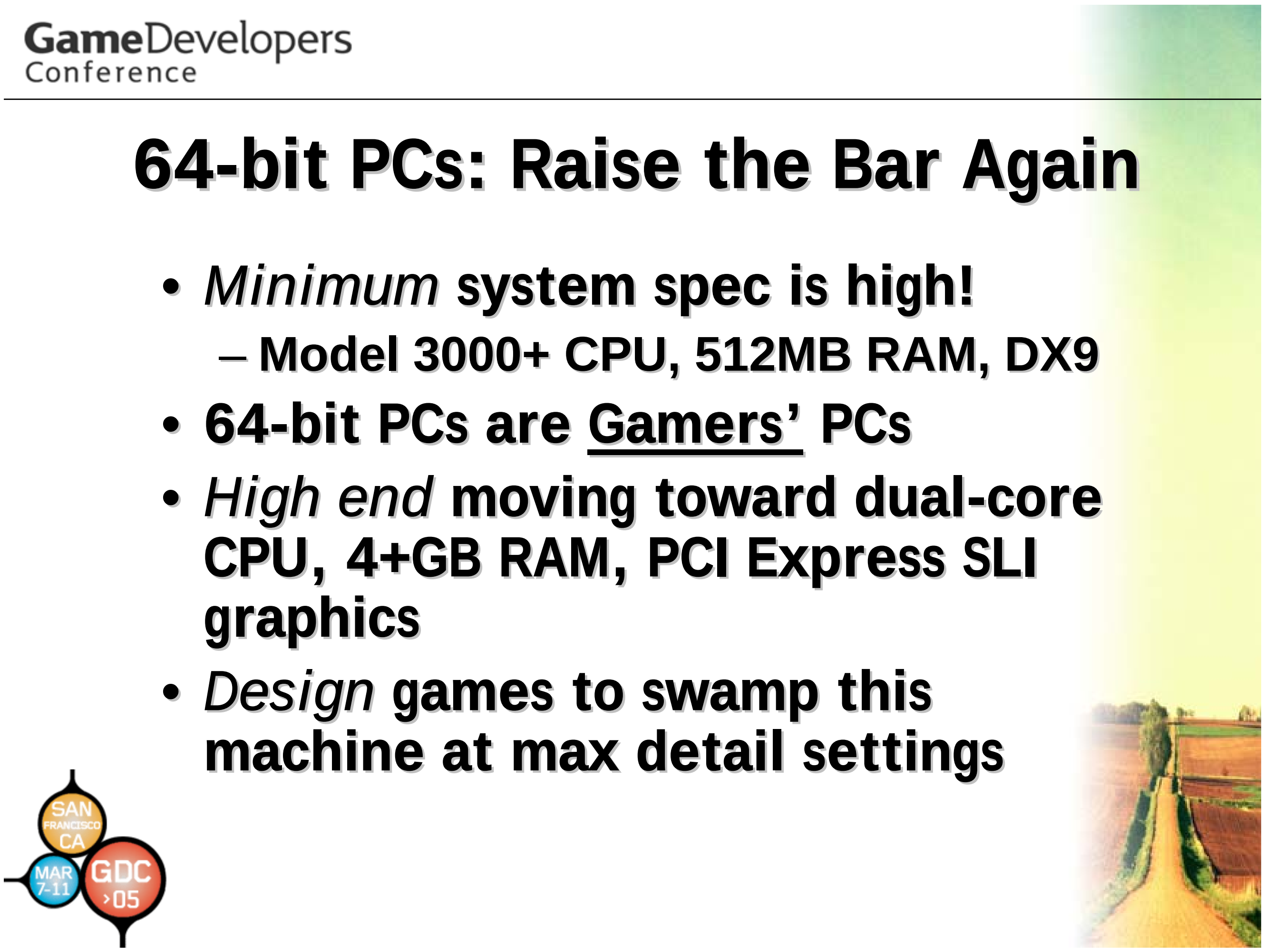


GameDevelopers Conference

Maximizing Performance of PC Games on 64-bit Platforms

Michael Wall
Senior Member of Technical Staff
Advanced Micro Devices, Inc.

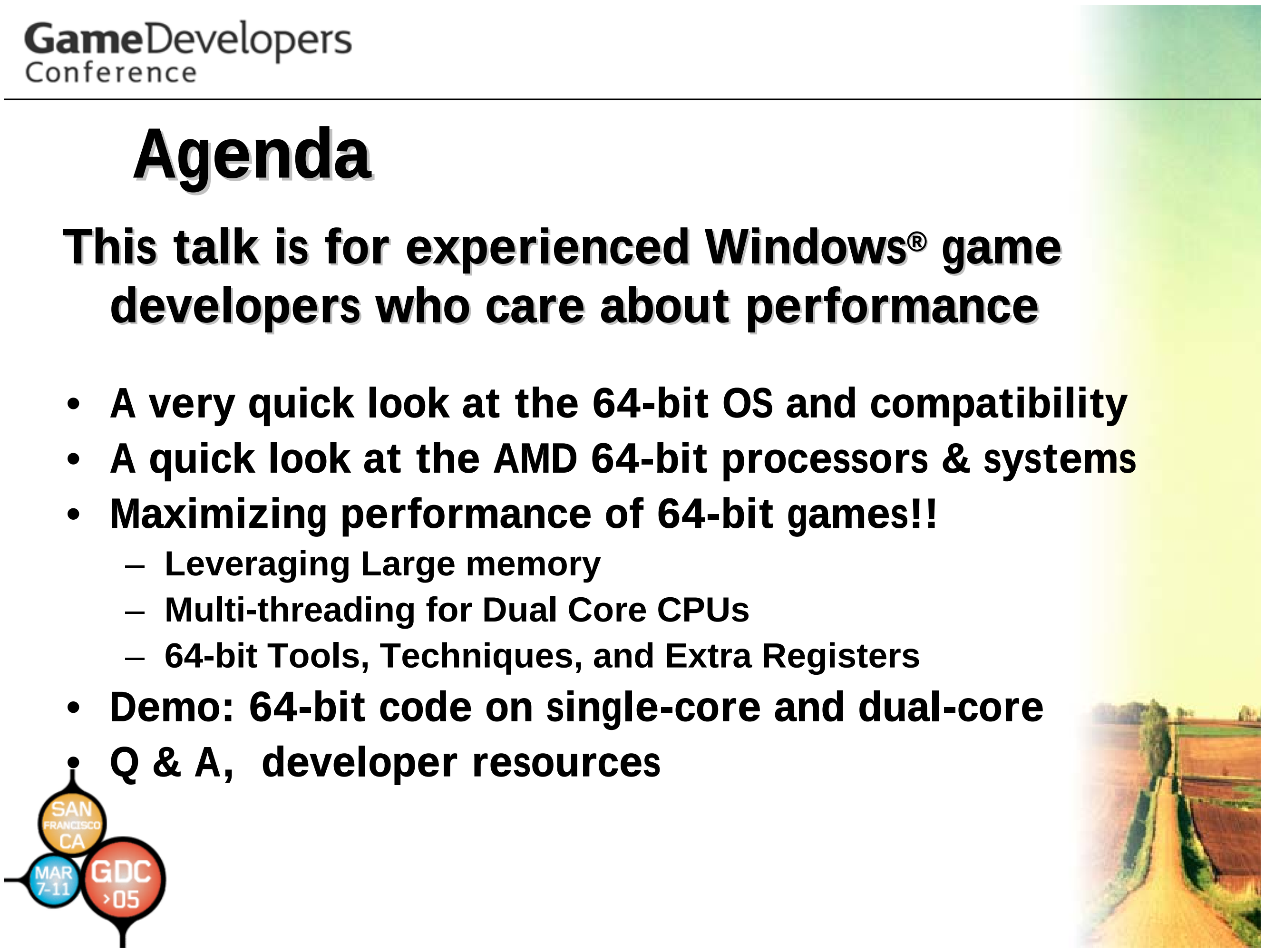


GameDevelopers Conference

64-bit PCs: Raise the Bar Again

- *Minimum* system spec is high!
  - Model 3000+ CPU, 512MB RAM, DX9
- 64-bit PCs are <u>Gamers'</u> PCs
- *High end* moving toward dual-core CPU, 4+GB RAM, PCI Express SLI graphics
- *Design* games to swamp this machine at max detail settings

SAN FRANCISCO CA MAR 7-11 GDC ›05



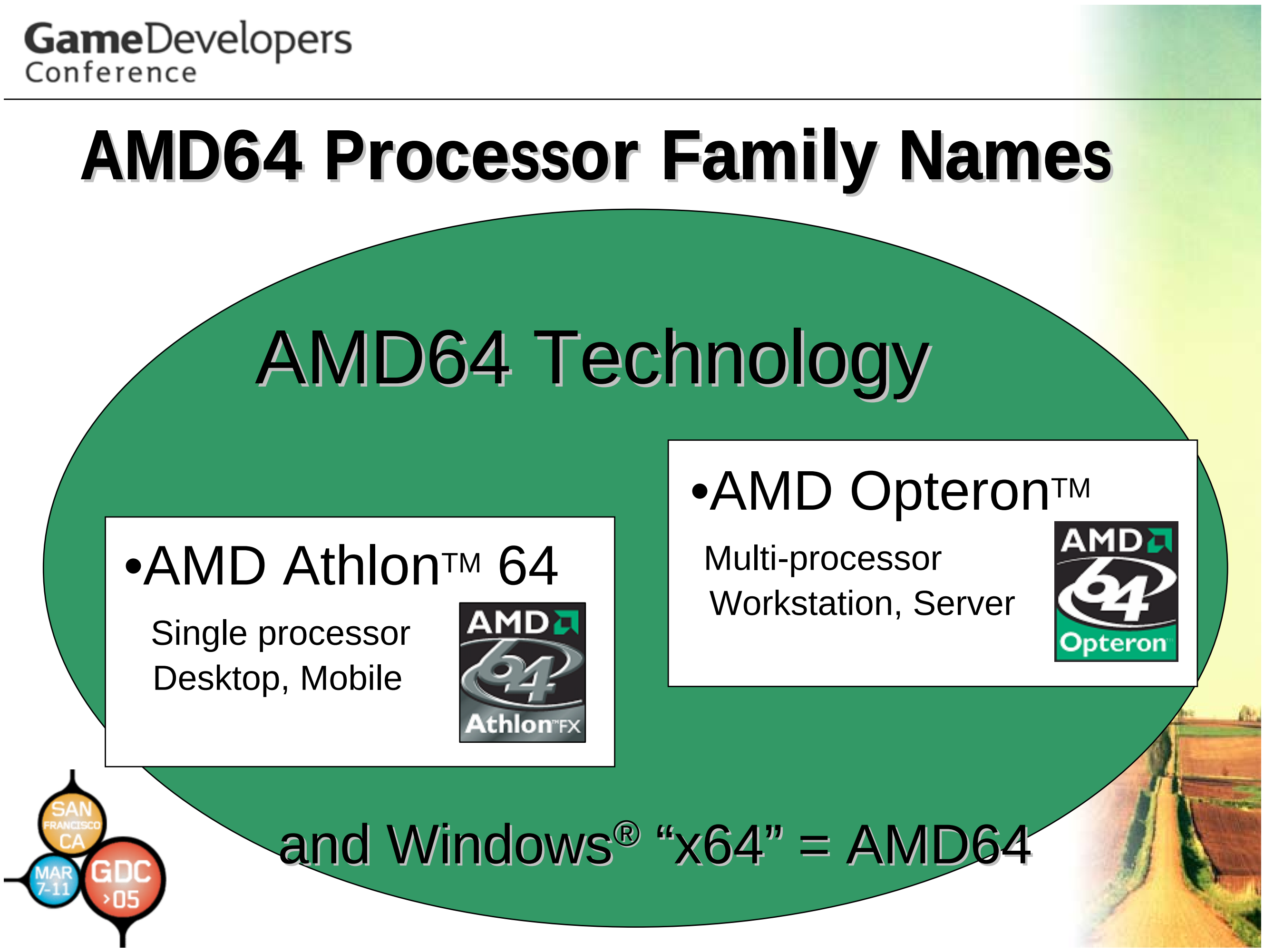
GameDevelopers Conference

Agenda

This talk is for experienced Windows® game developers who care about performance

- A very quick look at the 64-bit OS and compatibility
- A quick look at the AMD 64-bit processors & systems
- Maximizing performance of 64-bit games!!
  - Leveraging Large memory
  - Multi-threading for Dual Core CPUs
  - 64-bit Tools, Techniques, and Extra Registers
- Demo: 64-bit code on single-core and dual-core
- Q & A, developer resources

SAN FRANCISCO CA MAR 7-11 GDC ›05



GameDevelopers Conference

AMD64 Processor Family Names

AMD64 Technology

- AMD Athlon™ 64

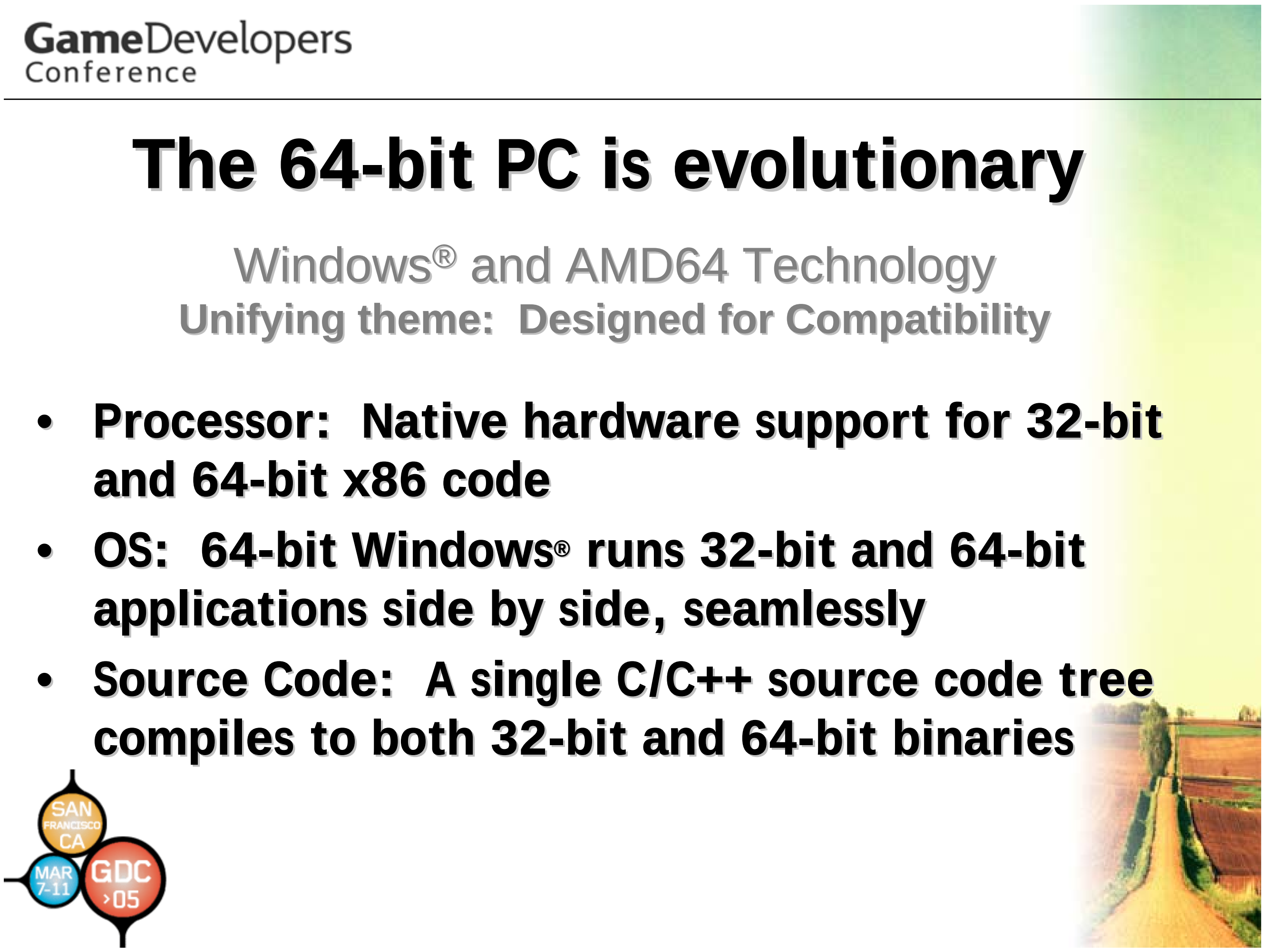
  Single processor
  Desktop, Mobile

- AMD Opteron™

  Multi-processor
  Workstation, Server

and Windows® "x64" = AMD64

AMD 64 Athlon FX
AMD 64 Opteron
SAN FRANCISCO CA
MAR 7-11
GDC ›05



GameDevelopers Conference

The 64-bit PC is evolutionary

Windows® and AMD64 Technology

Unifying theme: Designed for Compatibility

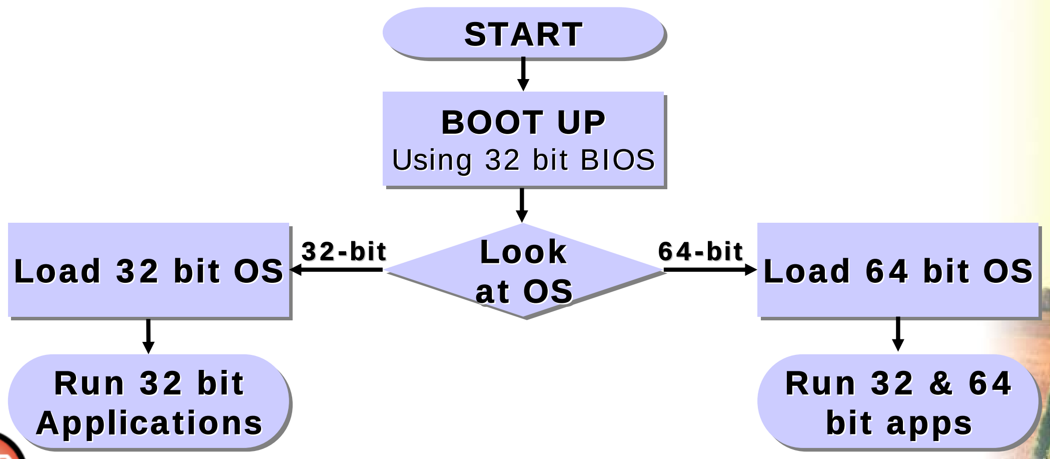
- **Processor: Native hardware support for 32-bit and 64-bit x86 code**
- **OS: 64-bit Windows® runs 32-bit and 64-bit applications side by side, seamlessly**
- **Source Code: A single C/C++ source code tree compiles to both 32-bit and 64-bit binaries**

SAN FRANCISCO CA MAR 7-11 GDC ›05

GameDevelopers Conference

Windows® for x64-based Systems
32-bit and 64-bit on a single platform

An AMD64-based Processor can run both 32- and 64-bit Windows® operating systems



SAN FRANCISCO CA MAR 7-11 GDC ›05

GameDevelopers Conference

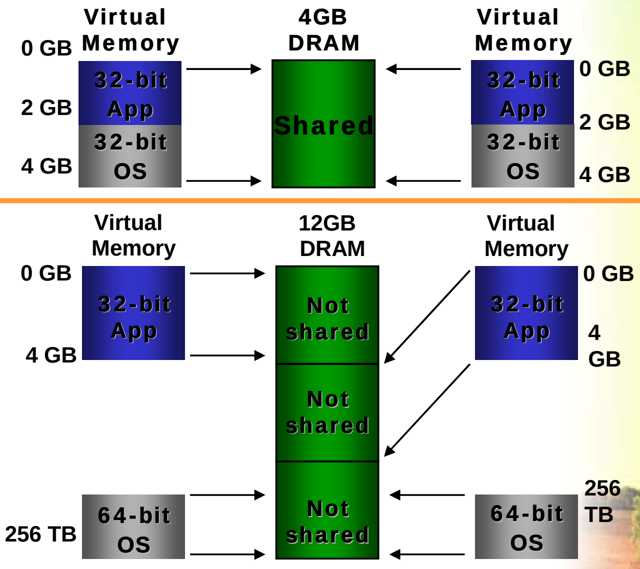
Why the 64-bit Windows® OS Benefits 32-bit Applications

32-bit x86 system

- OS and applications share virtual and physical memory
- Results in a lot of paging of info in and out of memory
- Limits the size of files and datasets

x64-based System

- OS uses Virtual Memory space outside range of 32-bits
- Application has exclusive use of virtual memory space
- OS can allocate each application dedicated portions of physical memory
- Reduces the amount of paging
- Supports larger file and dataset sizes



SAN FRANCISCO CA MAR 7-11 GDC ›05

GameDevelopers Conference

AMD64 Technology: Unique Architectural Advantages

Integrated Memory Controller

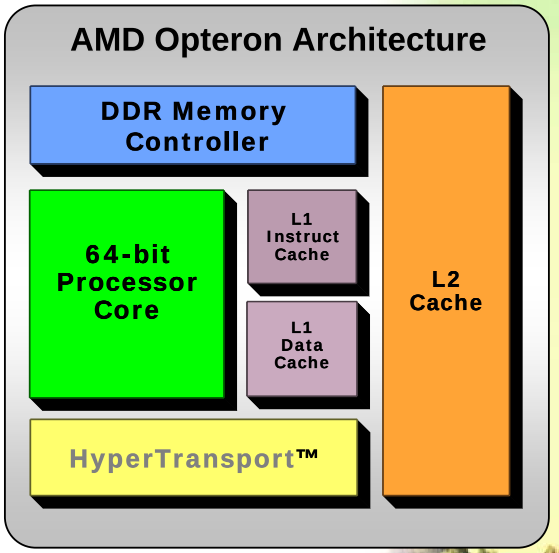
- Memory is directly attached to the processor providing high bandwidth, low latency access

64-bit Processor Core

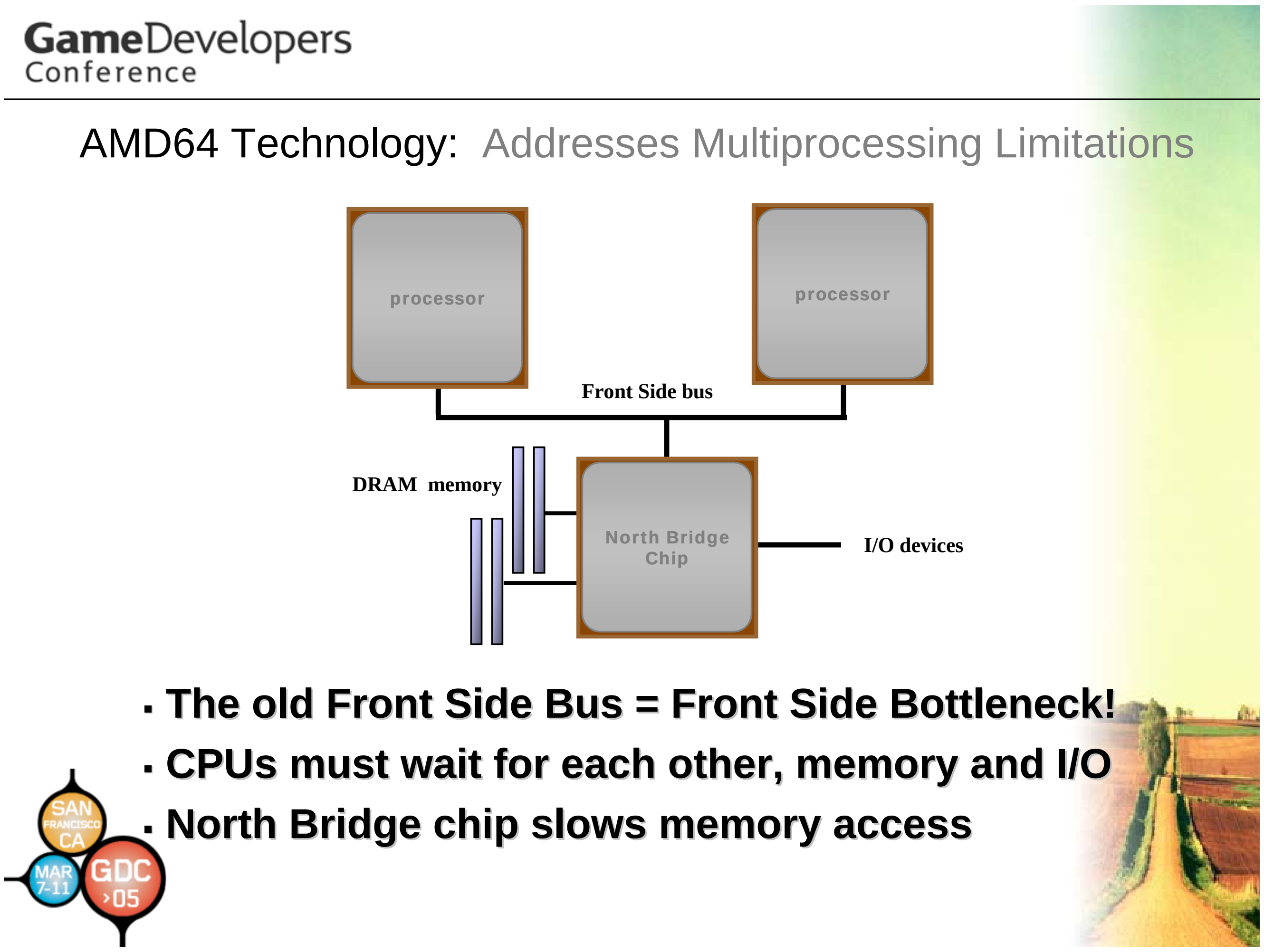
- Compatible with existing x86 applications while providing 64-bit memory addressing capabilities

HyperTransport Technology

- Provides higher bandwidth for efficient I/O activities and a glueless approach to multiprocessing

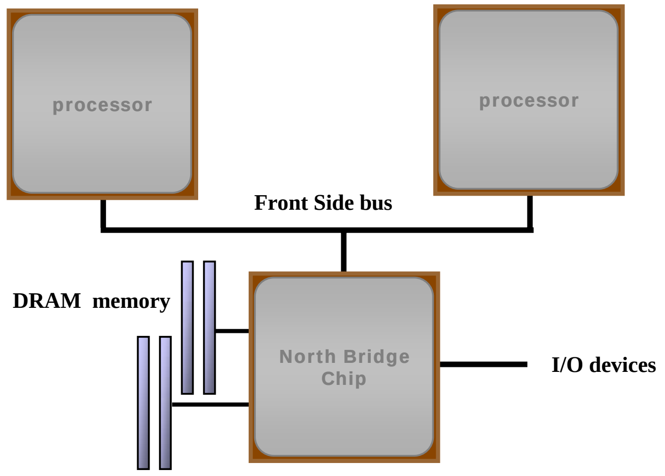


SAN FRANCISCO CA MAR 7-11 GDC ›05

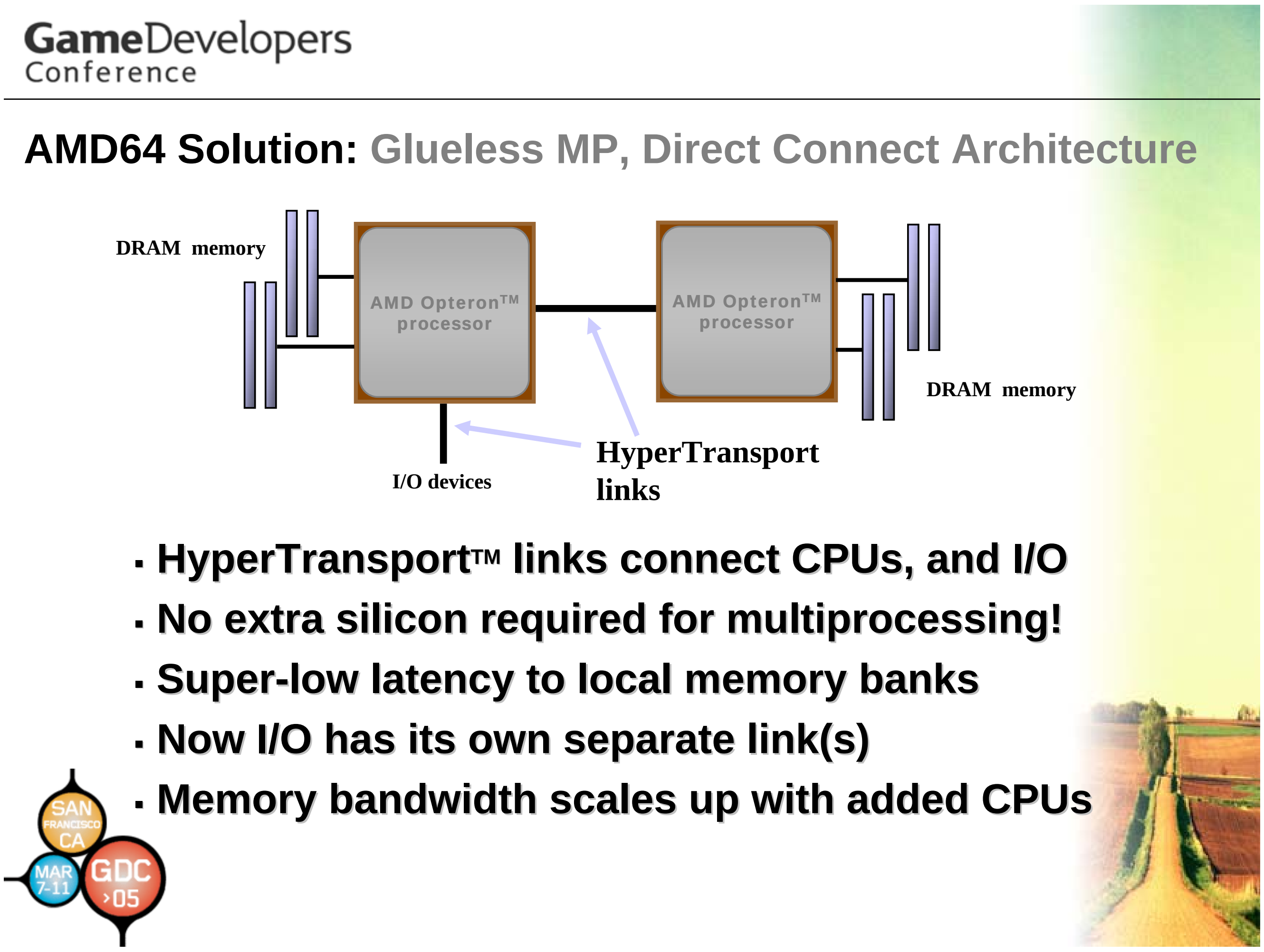


GameDevelopers Conference

AMD64 Technology: Addresses Multiprocessing Limitations

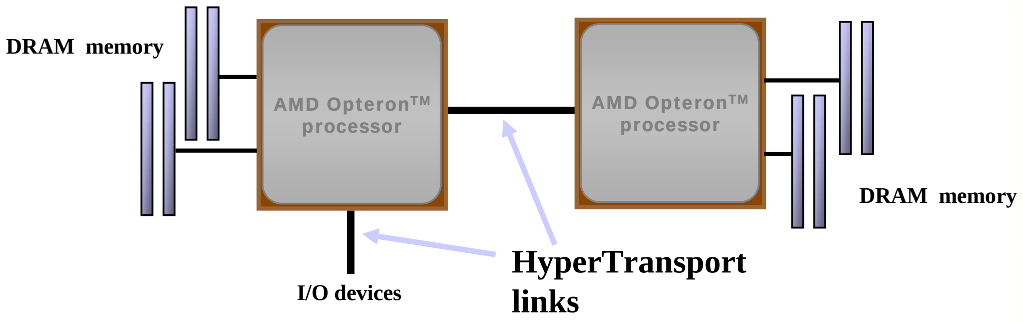


- The old Front Side Bus = Front Side Bottleneck!
- CPUs must wait for each other, memory and I/O
- North Bridge chip slows memory access

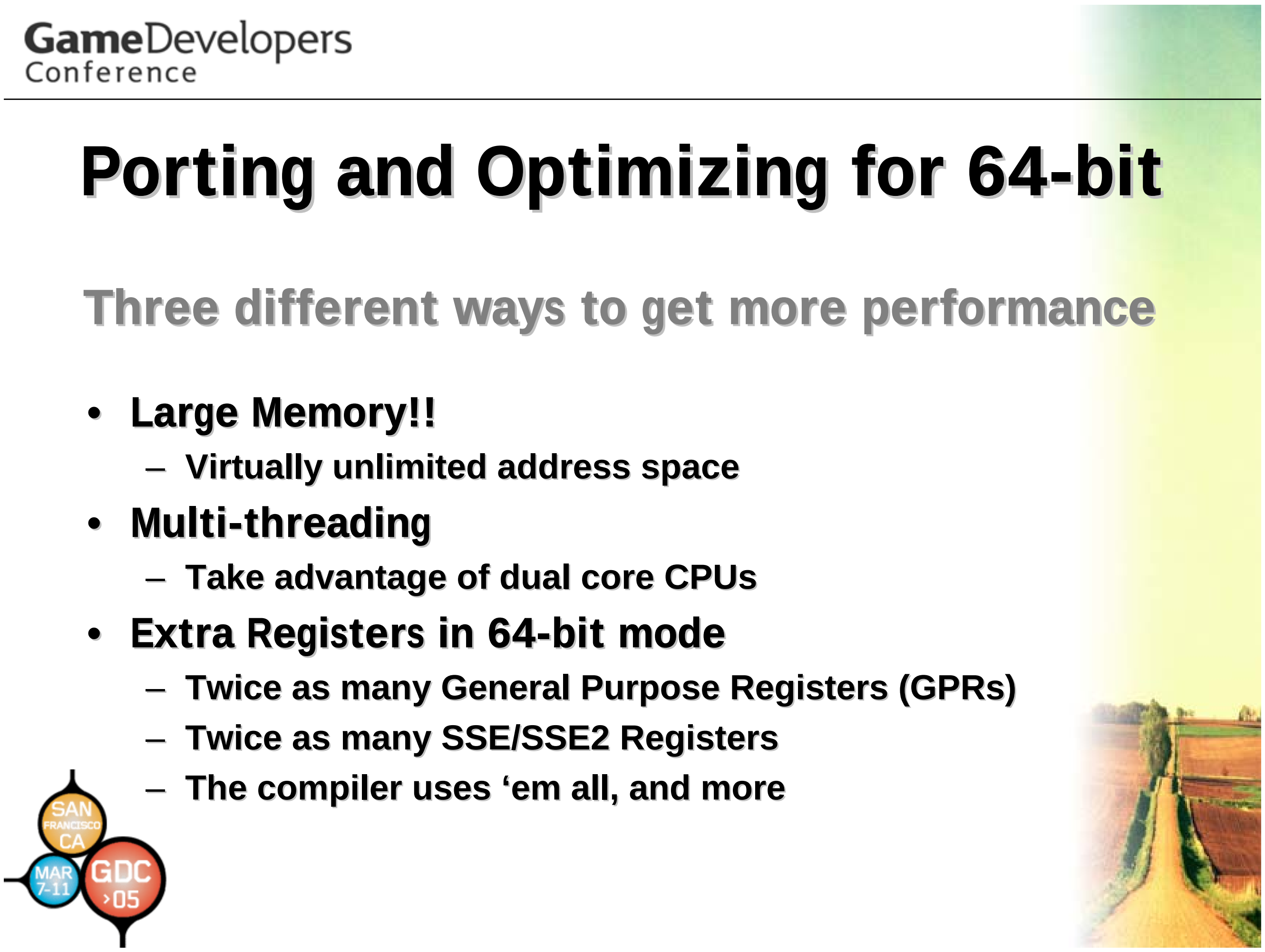


GameDevelopers Conference

AMD64 Solution: Glueless MP, Direct Connect Architecture



- HyperTransport™ links connect CPUs, and I/O
- No extra silicon required for multiprocessing!
- Super-low latency to local memory banks
- Now I/O has its own separate link(s)
- Memory bandwidth scales up with added CPUs



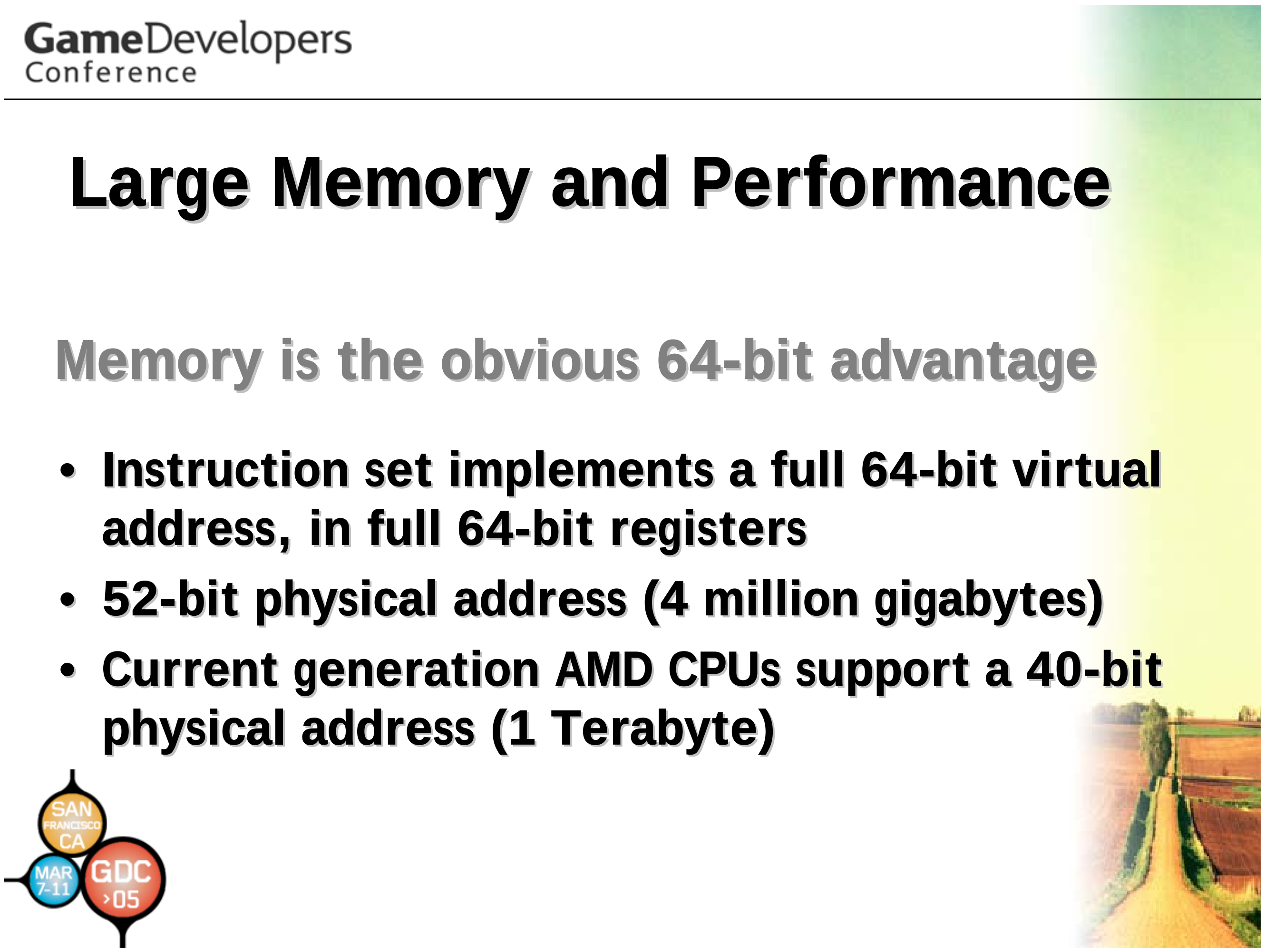
GameDevelopers Conference

Porting and Optimizing for 64-bit

Three different ways to get more performance

- Large Memory!!
  - Virtually unlimited address space
- Multi-threading
  - Take advantage of dual core CPUs
- Extra Registers in 64-bit mode
  - Twice as many General Purpose Registers (GPRs)
  - Twice as many SSE/SSE2 Registers
  - The compiler uses ‘em all, and more

SAN FRANCISCO CA MAR 7-11 GDC ›05



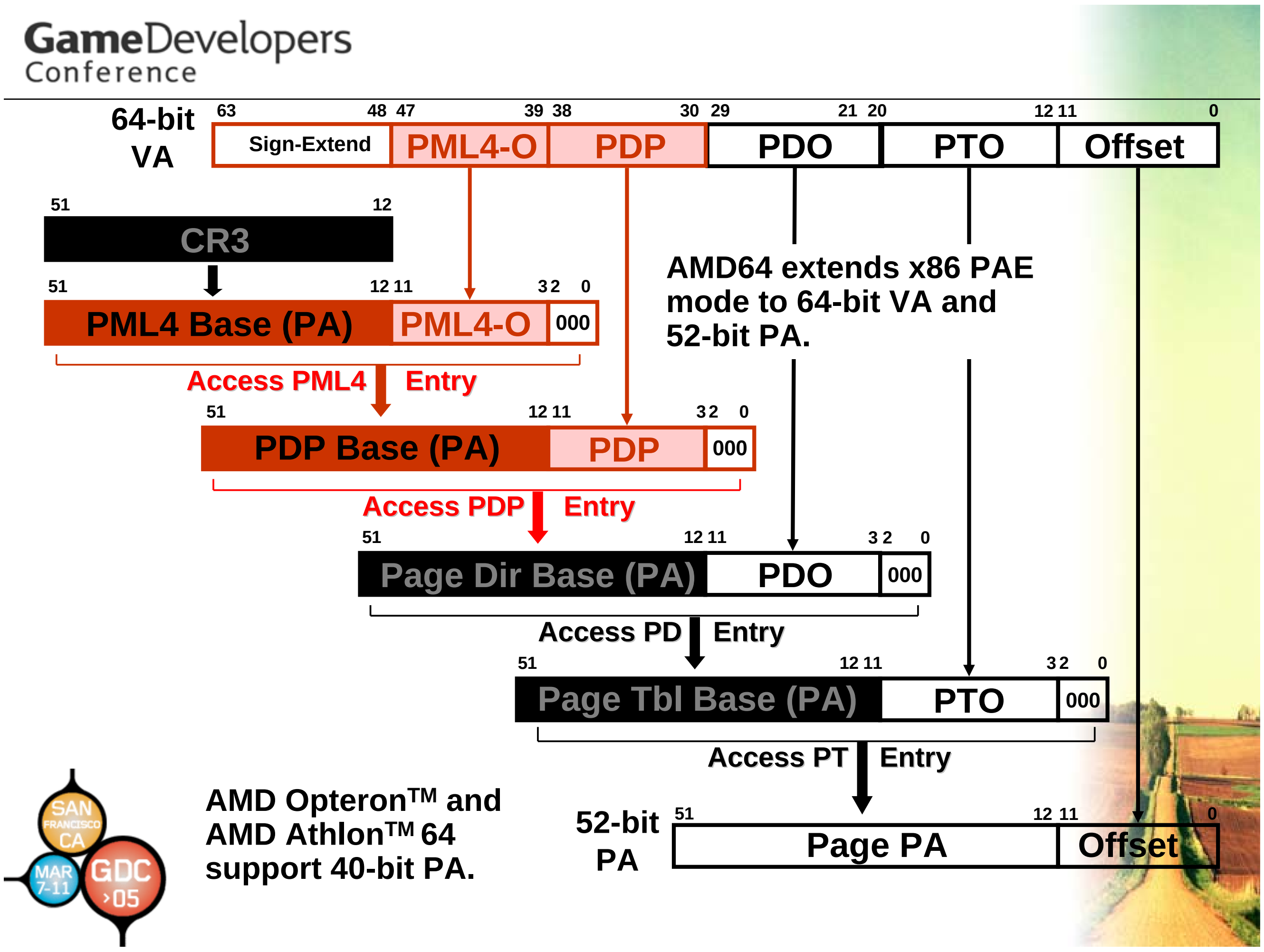
GameDevelopers Conference

Large Memory and Performance

Memory is the obvious 64-bit advantage

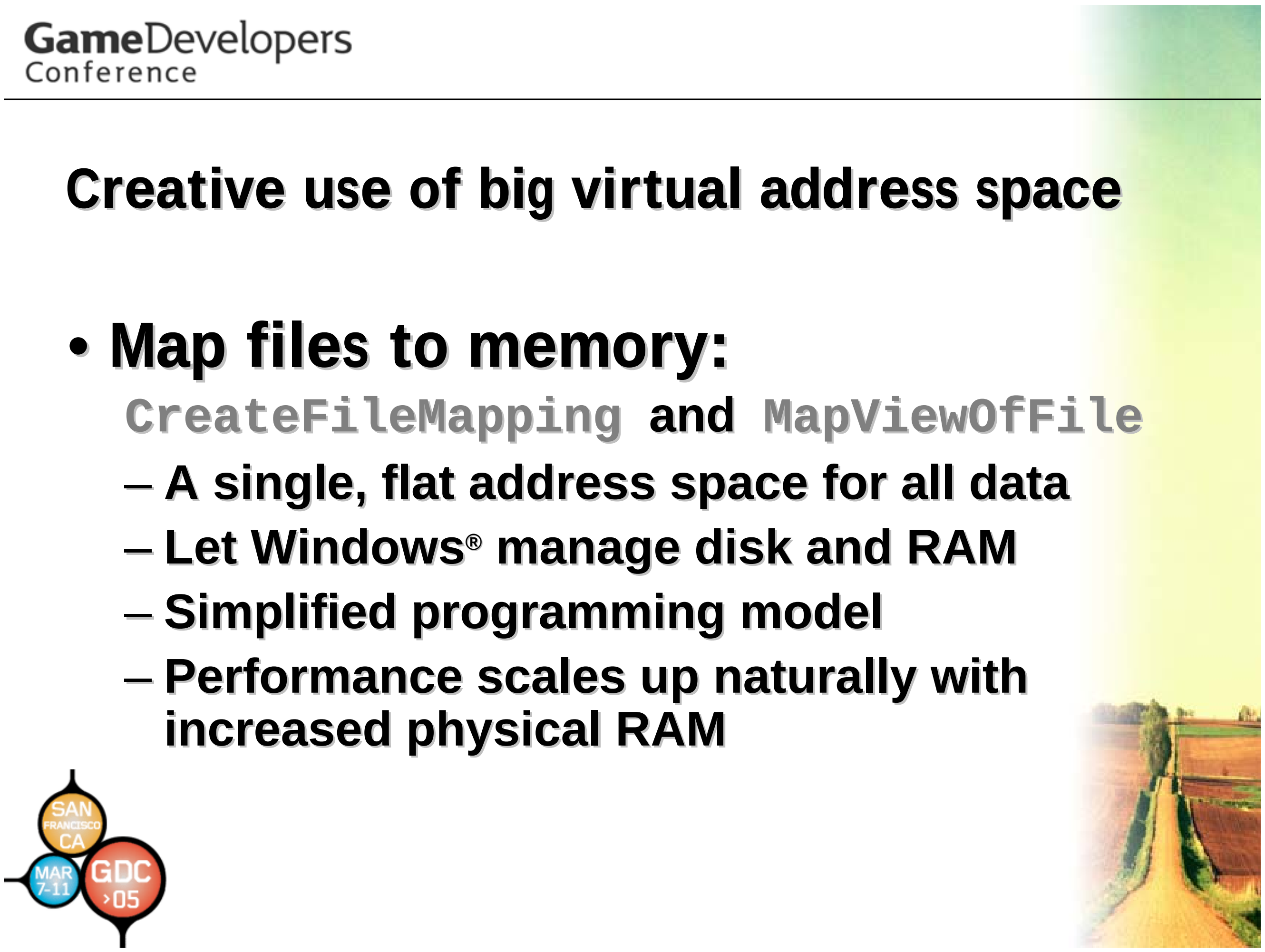
- Instruction set implements a full 64-bit virtual address, in full 64-bit registers
- 52-bit physical address (4 million gigabytes)
- Current generation AMD CPUs support a 40-bit physical address (1 Terabyte)

SAN FRANCISCO CA
MAR 7-11
GDC ›05



AMD64 extends x86 PAE mode to 64-bit VA and 52-bit PA.

AMD Opteron™ and AMD Athlon™ 64 support 40-bit PA.

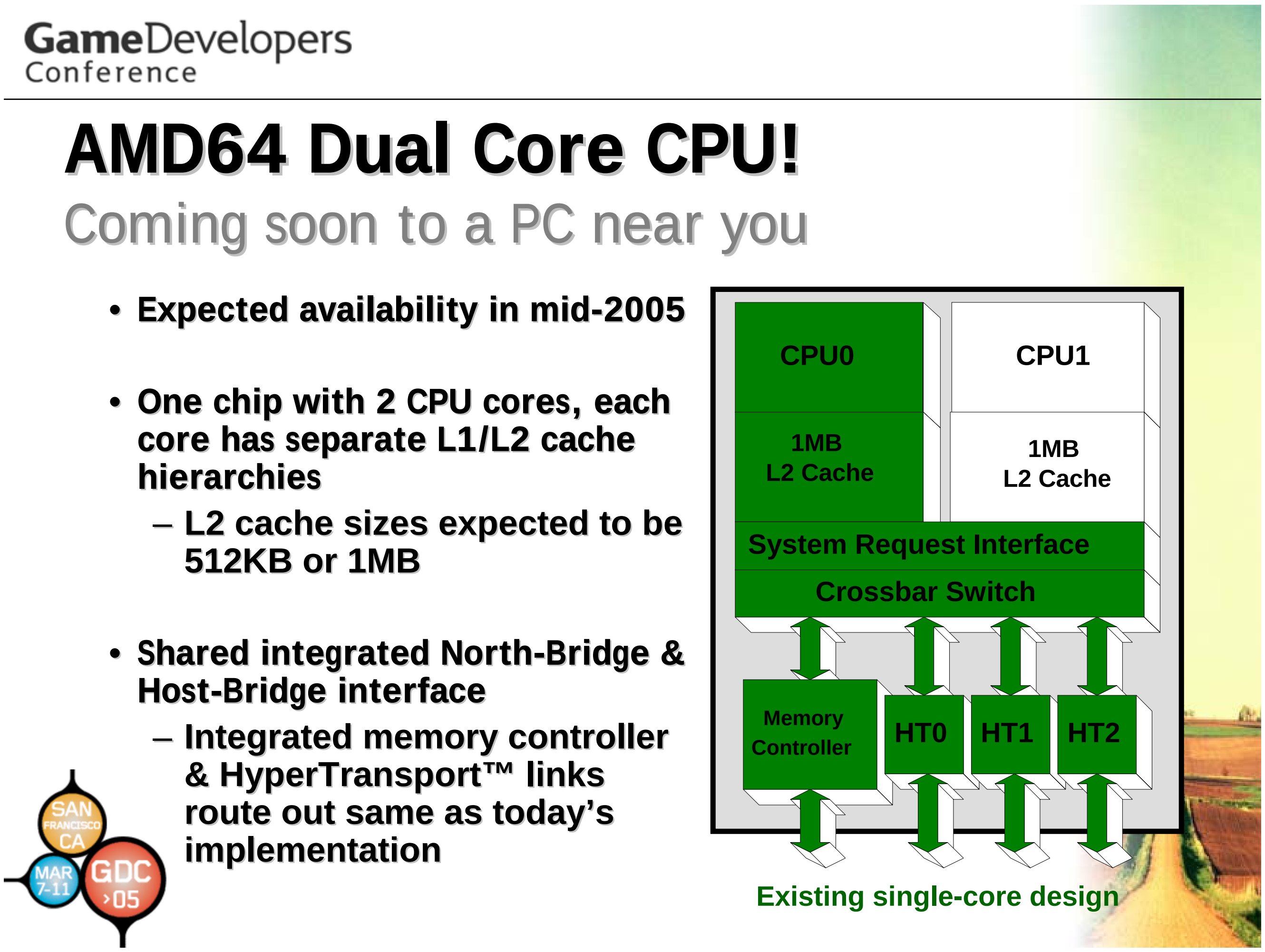


GameDevelopers Conference

Creative use of big virtual address space

- **Map files to memory:**
  `CreateFileMapping` and `MapViewOfFile`
  - A single, flat address space for all data
  - Let Windows® manage disk and RAM
  - Simplified programming model
  - Performance scales up naturally with increased physical RAM

SAN FRANCISCO CA MAR 7-11 GDC ›05



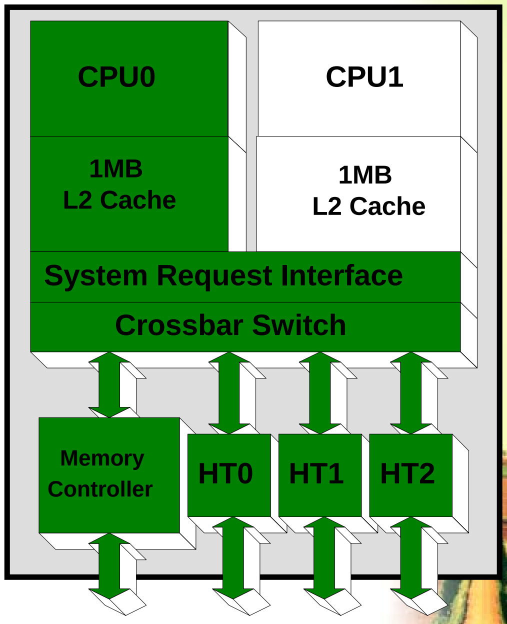
GameDevelopers Conference

AMD64 Dual Core CPU!

Coming soon to a PC near you

- Expected availability in mid-2005
- One chip with 2 CPU cores, each core has separate L1/L2 cache hierarchies
  - L2 cache sizes expected to be 512KB or 1MB
- Shared integrated North-Bridge & Host-Bridge interface
  - Integrated memory controller & HyperTransport™ links route out same as today's implementation

SAN FRANCISCO CA MAR 7-11 GDC ›05



Existing single-core design

GameDevelopers Conference

AMD64 Dual-Core Physical Design

90nm

Approximately same die size as 130nm single-core AMD Opteron™ processor*

~205 million transistors*

95 watt power envelope

Fits into 90nm power infrastructure

940 Socket compatible

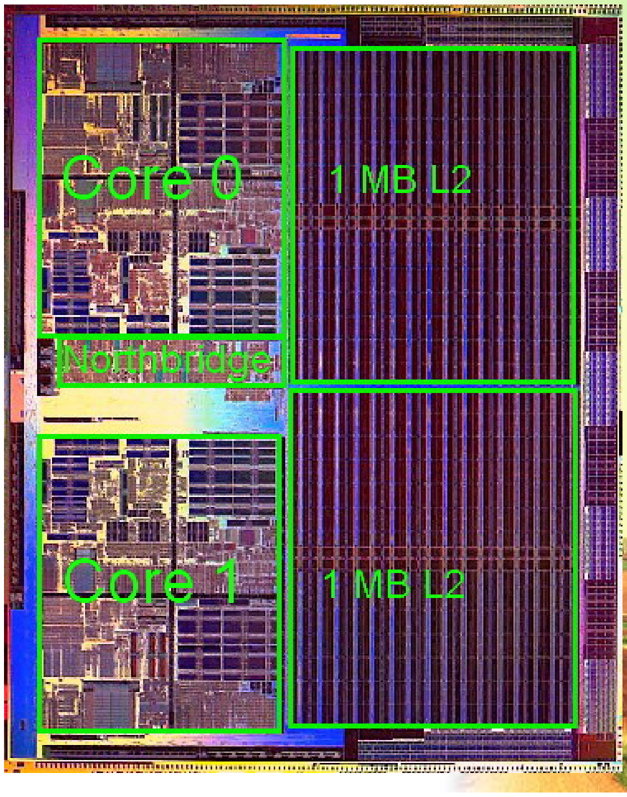
- **Opteron**

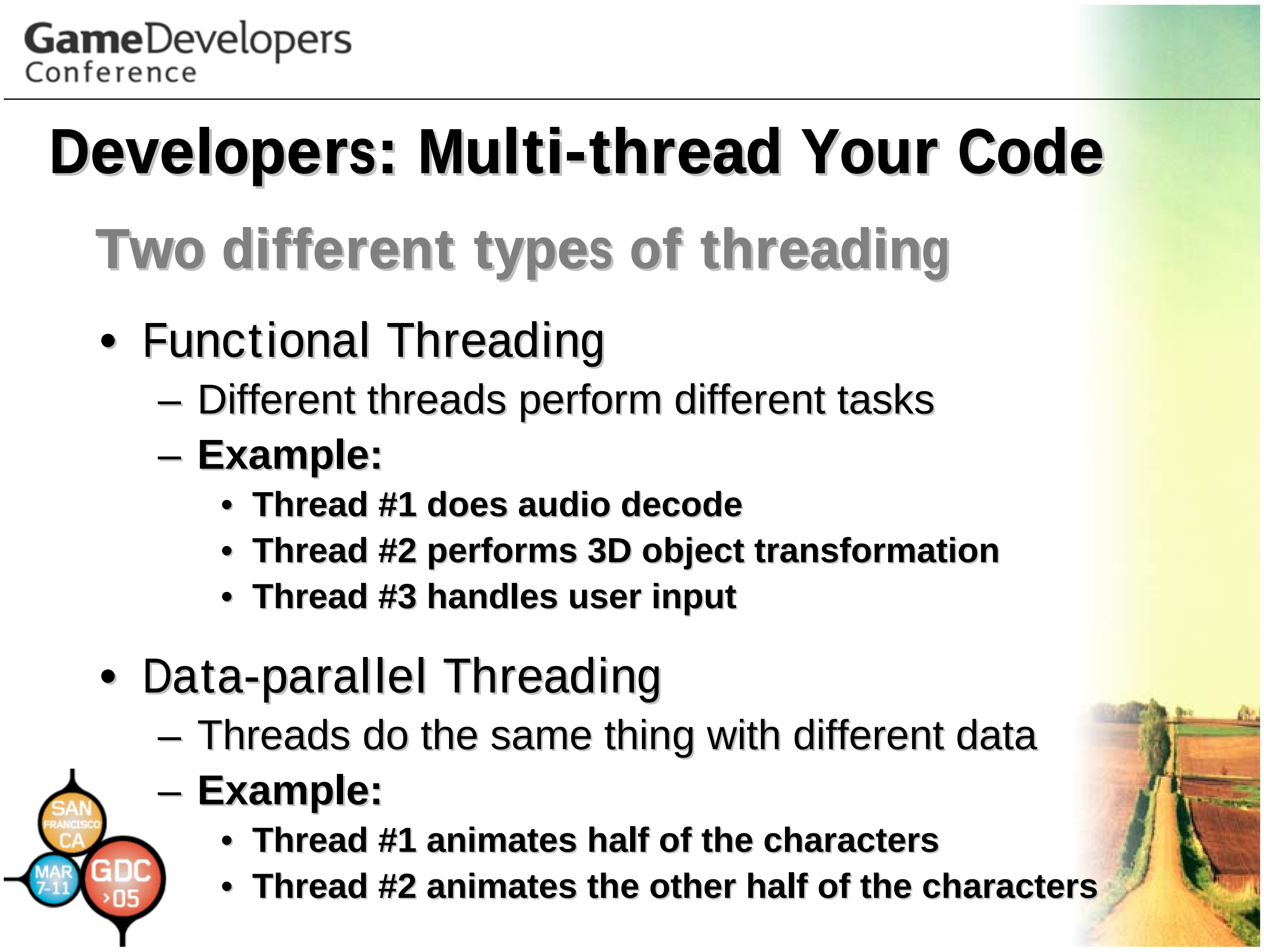
939 Socket compatible

- **Athlon 64**

SAN FRANCISCO CA MAR 7-11 GDC 05

**Based on current revisions of the design*





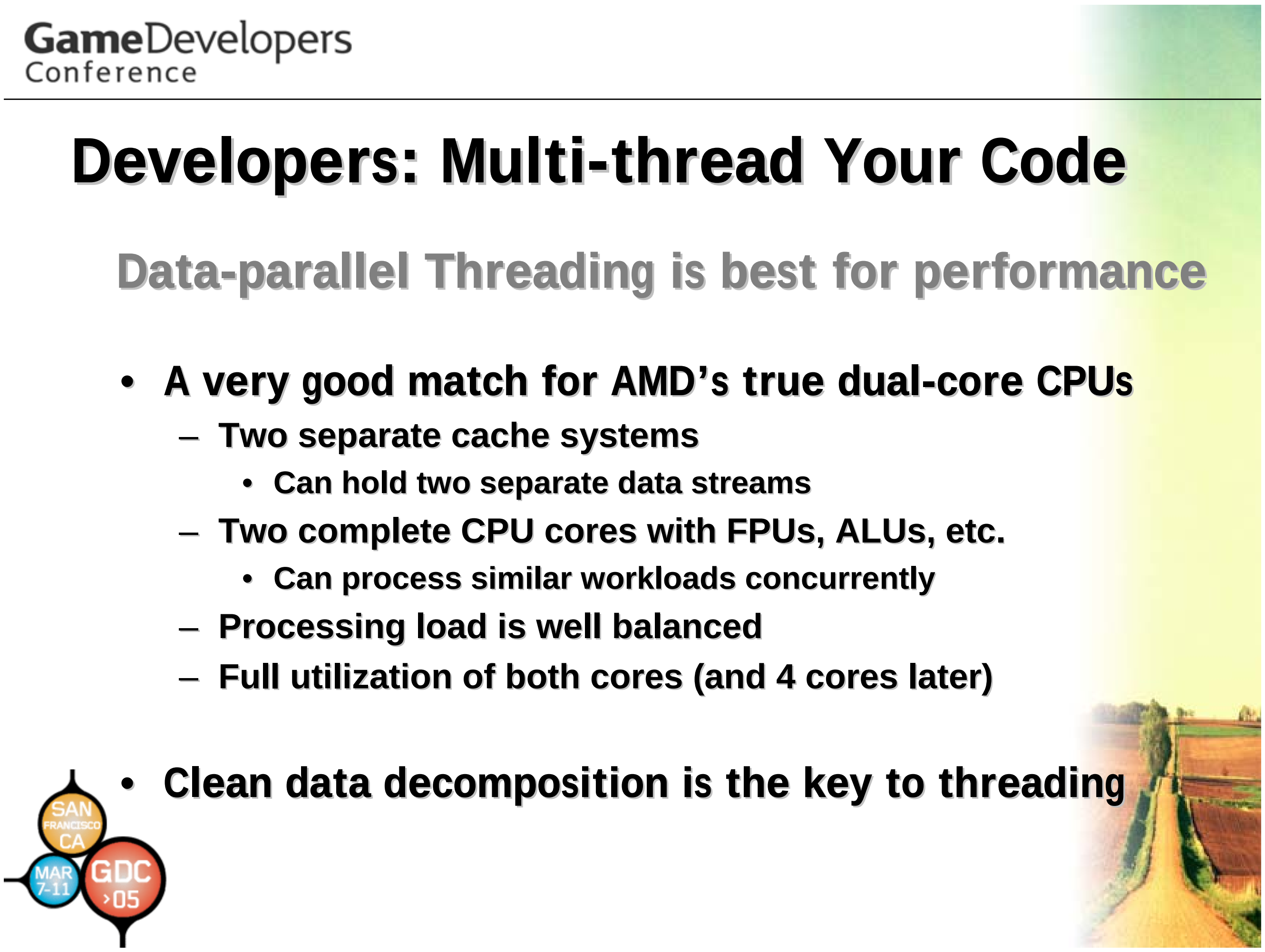
GameDevelopers Conference

Developers: Multi-thread Your Code

Two different types of threading

- Functional Threading
  - Different threads perform different tasks
  - **Example:**
    - **Thread #1 does audio decode**
    - **Thread #2 performs 3D object transformation**
    - **Thread #3 handles user input**
- Data-parallel Threading
  - Threads do the same thing with different data
  - **Example:**
    - **Thread #1 animates half of the characters**
    - **Thread #2 animates the other half of the characters**

SAN FRANCISCO CA MAR 7-11 GDC ›05



GameDevelopers Conference

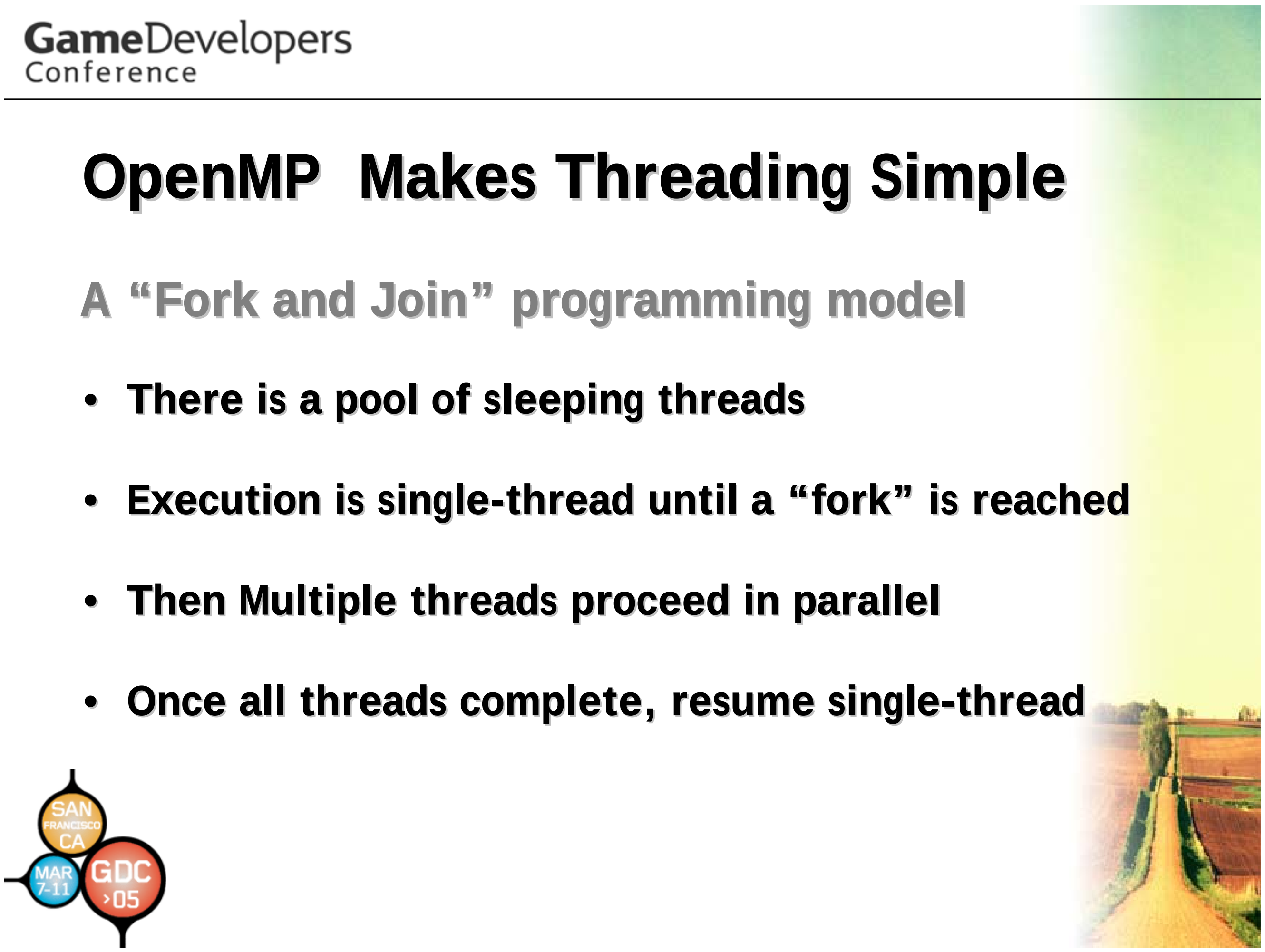
Developers: Multi-thread Your Code

Data-parallel Threading is best for performance

- **A very good match for AMD's true dual-core CPUs**
  - Two separate cache systems
    - Can hold two separate data streams
  - Two complete CPU cores with FPUs, ALUs, etc.
    - Can process similar workloads concurrently
  - Processing load is well balanced
  - Full utilization of both cores (and 4 cores later)

- **Clean data decomposition is the key to threading**

SAN FRANCISCO CA MAR 7-11 GDC ›05



GameDevelopers Conference

OpenMP Makes Threading Simple

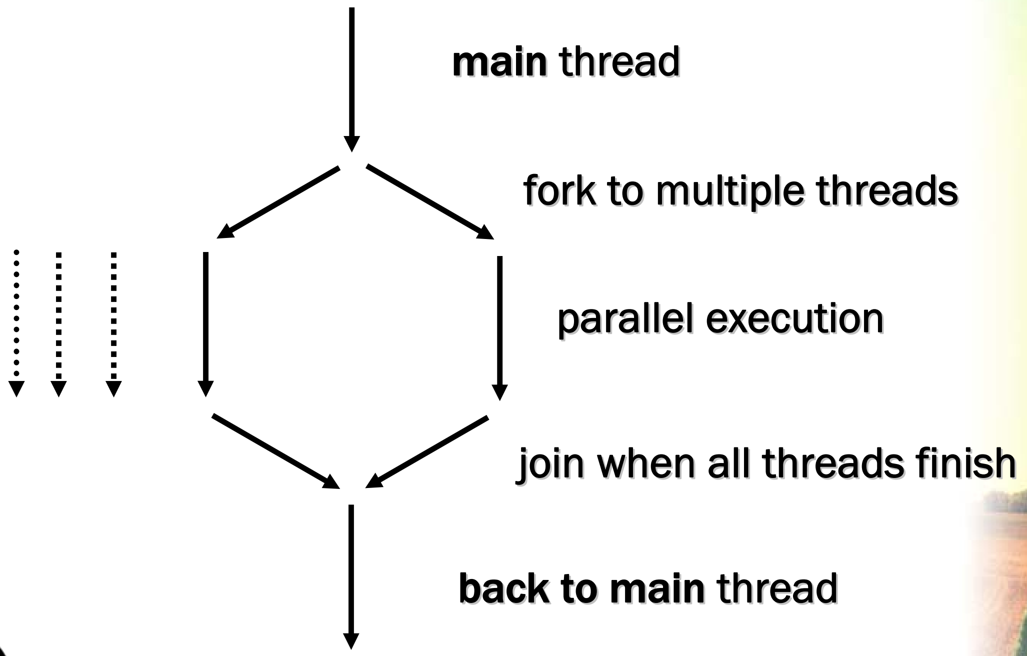
A “Fork and Join” programming model

- There is a pool of sleeping threads
- Execution is single-thread until a “fork” is reached
- Then Multiple threads proceed in parallel
- Once all threads complete, resume single-thread

SAN FRANCISCO CA MAR 7-11 GDC ›05

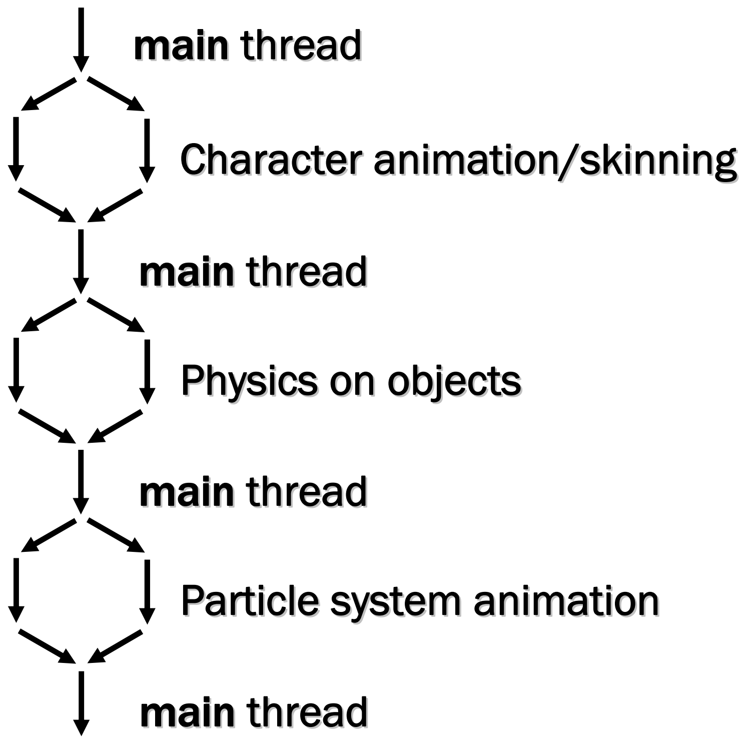
GameDevelopers Conference

Fork and Join Programming Model



GameDevelopers Conference

Thread Example: game pipeline (don’t laugh)



main thread

Character animation/skinning

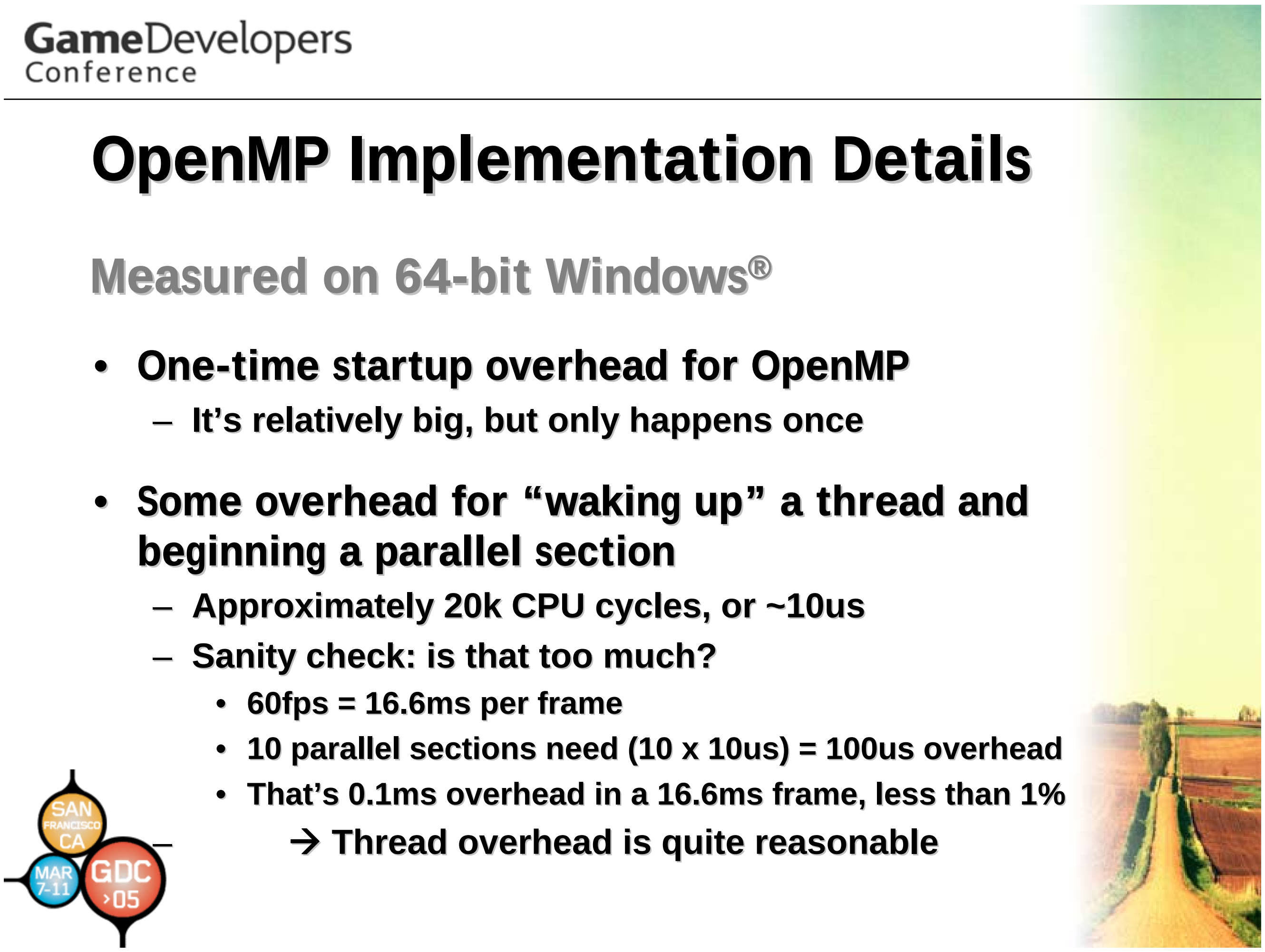
main thread

Physics on objects

main thread

Particle system animation

main thread



GameDevelopers Conference

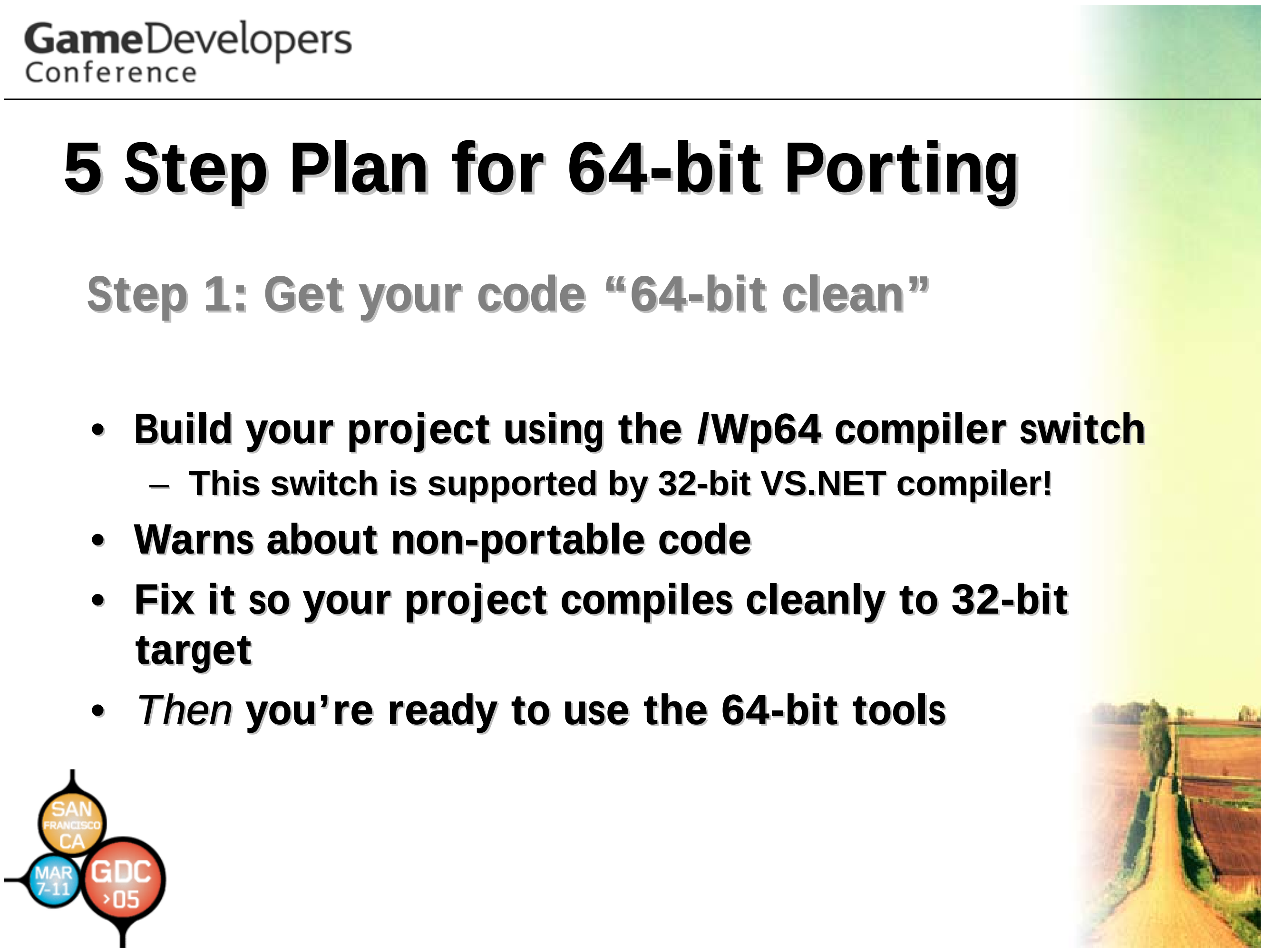
OpenMP Implementation Details

Measured on 64-bit Windows®

- One-time startup overhead for OpenMP
  - It's relatively big, but only happens once
- Some overhead for "waking up" a thread and beginning a parallel section
  - Approximately 20k CPU cycles, or ~10us
  - Sanity check: is that too much?
    - 60fps = 16.6ms per frame
    - 10 parallel sections need (10 x 10us) = 100us overhead
    - That's 0.1ms overhead in a 16.6ms frame, less than 1%

→ Thread overhead is quite reasonable

SAN FRANCISCO CA MAR 7-11 GDC ›05



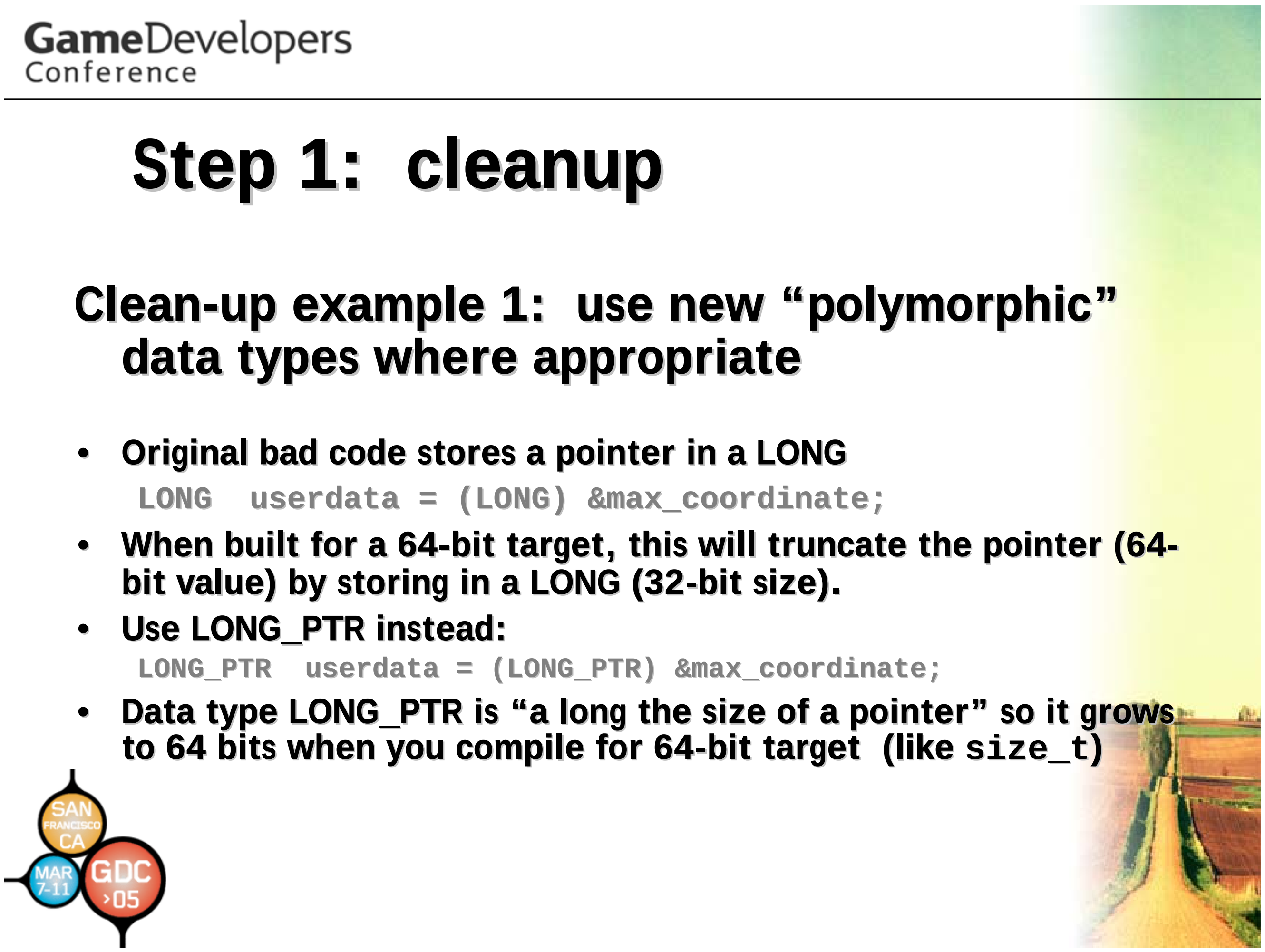
GameDevelopers Conference

5 Step Plan for 64-bit Porting

Step 1: Get your code "64-bit clean"

- Build your project using the /Wp64 compiler switch
  - This switch is supported by 32-bit VS.NET compiler!
- Warns about non-portable code
- Fix it so your project compiles cleanly to 32-bit target
- *Then* you're ready to use the 64-bit tools

SAN FRANCISCO CA MAR 7-11 GDC ›05



GameDevelopers Conference

Step 1: cleanup

Clean-up example 1: use new “polymorphic” data types where appropriate

- Original bad code stores a pointer in a LONG

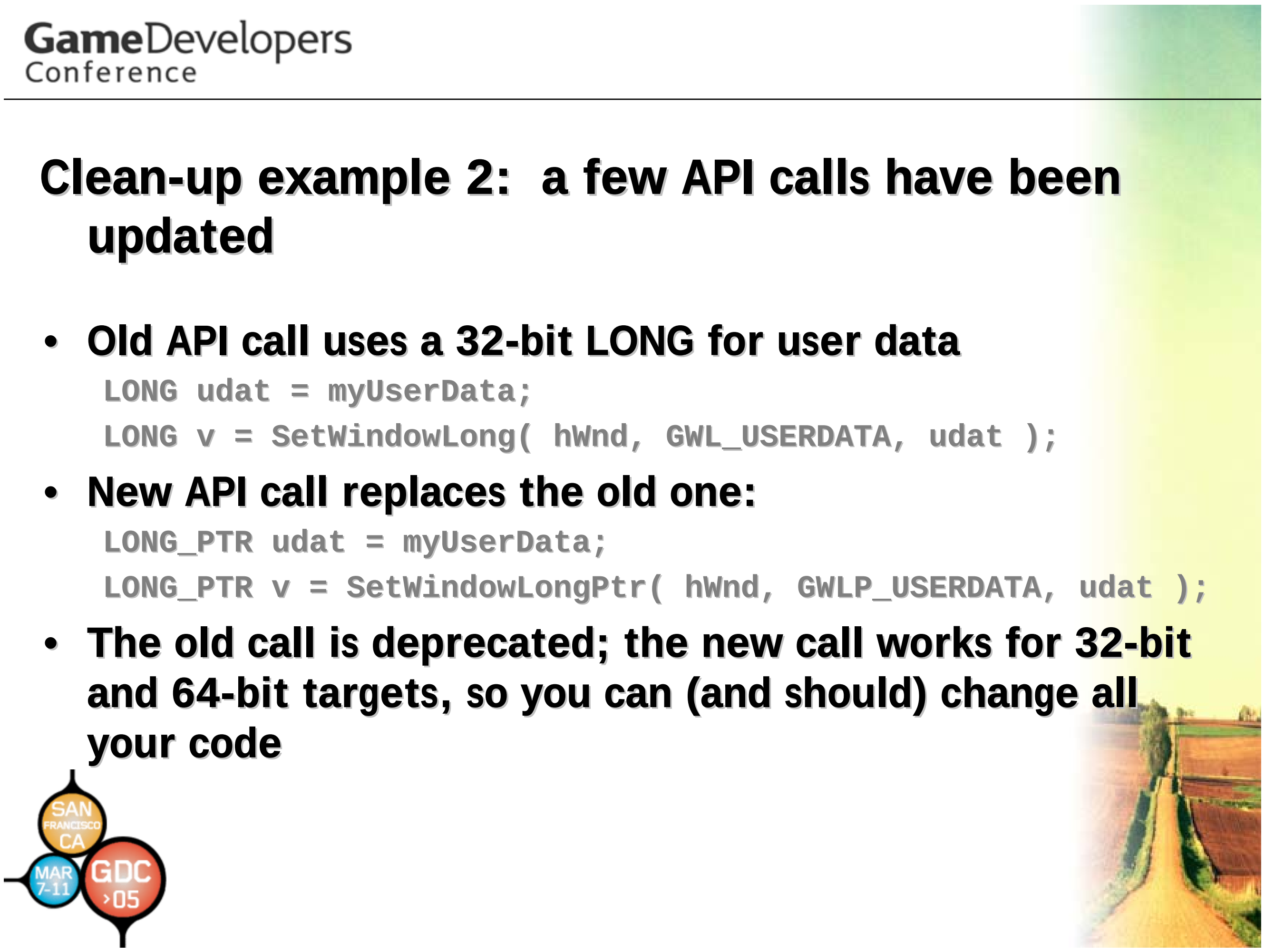
```
LONG  userdata = (LONG) &max_coordinate;
```

- When built for a 64-bit target, this will truncate the pointer (64-bit value) by storing in a LONG (32-bit size).
- Use LONG_PTR instead:

```
LONG_PTR  userdata = (LONG_PTR) &max_coordinate;
```

- Data type LONG_PTR is “a long the size of a pointer” so it grows to 64 bits when you compile for 64-bit target (like `size_t`)

SAN FRANCISCO CA MAR 7-11 GDC ›05



GameDevelopers Conference

Clean-up example 2: a few API calls have been updated

- Old API call uses a 32-bit LONG for user data

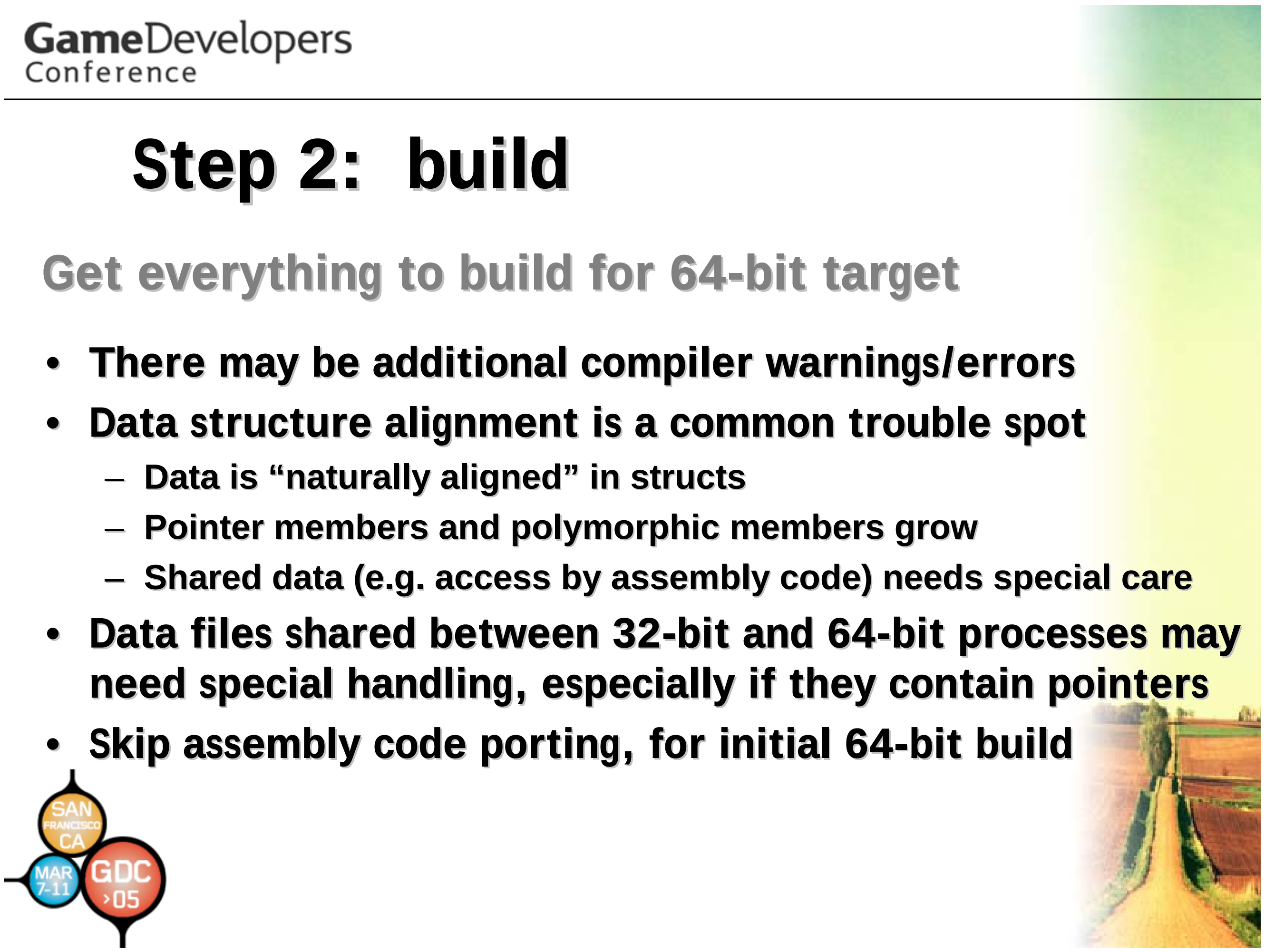
```
LONG udat = myUserData;
LONG v = SetWindowLong( hWnd, GWL_USERDATA, udat );
```

- New API call replaces the old one:

```
LONG_PTR udat = myUserData;
LONG_PTR v = SetWindowLongPtr( hWnd, GWLP_USERDATA, udat );
```

- The old call is deprecated; the new call works for 32-bit and 64-bit targets, so you can (and should) change all your code

SAN FRANCISCO CA MAR 7-11 GDC ›05



GameDevelopers Conference

Step 2: build

Get everything to build for 64-bit target

- There may be additional compiler warnings/errors
- Data structure alignment is a common trouble spot
  - Data is "naturally aligned" in structs
  - Pointer members and polymorphic members grow
  - Shared data (e.g. access by assembly code) needs special care
- Data files shared between 32-bit and 64-bit processes may need special handling, especially if they contain pointers
- Skip assembly code porting, for initial 64-bit build

SAN FRANCISCO CA MAR 7-11 GDC ›05

GameDevelopers Conference

Step 3: measure and analyze!

Benchmark and profile your 64-bit code

- **Performance profile may differ from old 32-bit build**
  - There are new drivers…
  - and there are new optimized libraries…
  - and a new 64-bit optimizing compiler…
  - all running on new CPU micro-architecture.

Focus optimization efforts on real, measured hot spots!

You may be surprised by where your code spends time

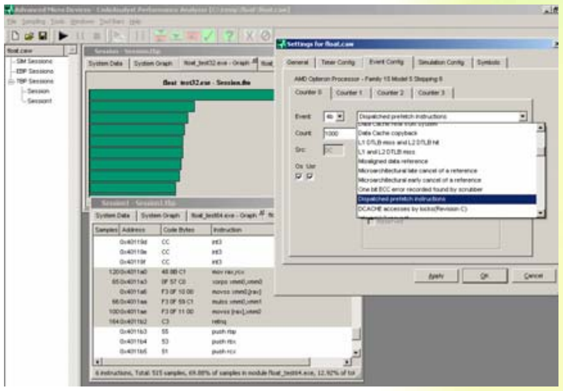
SAN FRANCISCO CA MAR 7-11 GDC ›05

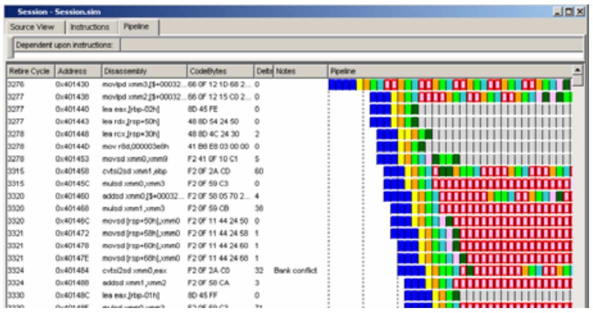
GameDevelopers Conference

Step 3: measure and analyze

Use AMD CodeAnalyst to profile your 64-bit code!!

- **Timer-based & event-based profiler**





Session - Session.sim

Source View | Instructions | Pipeline

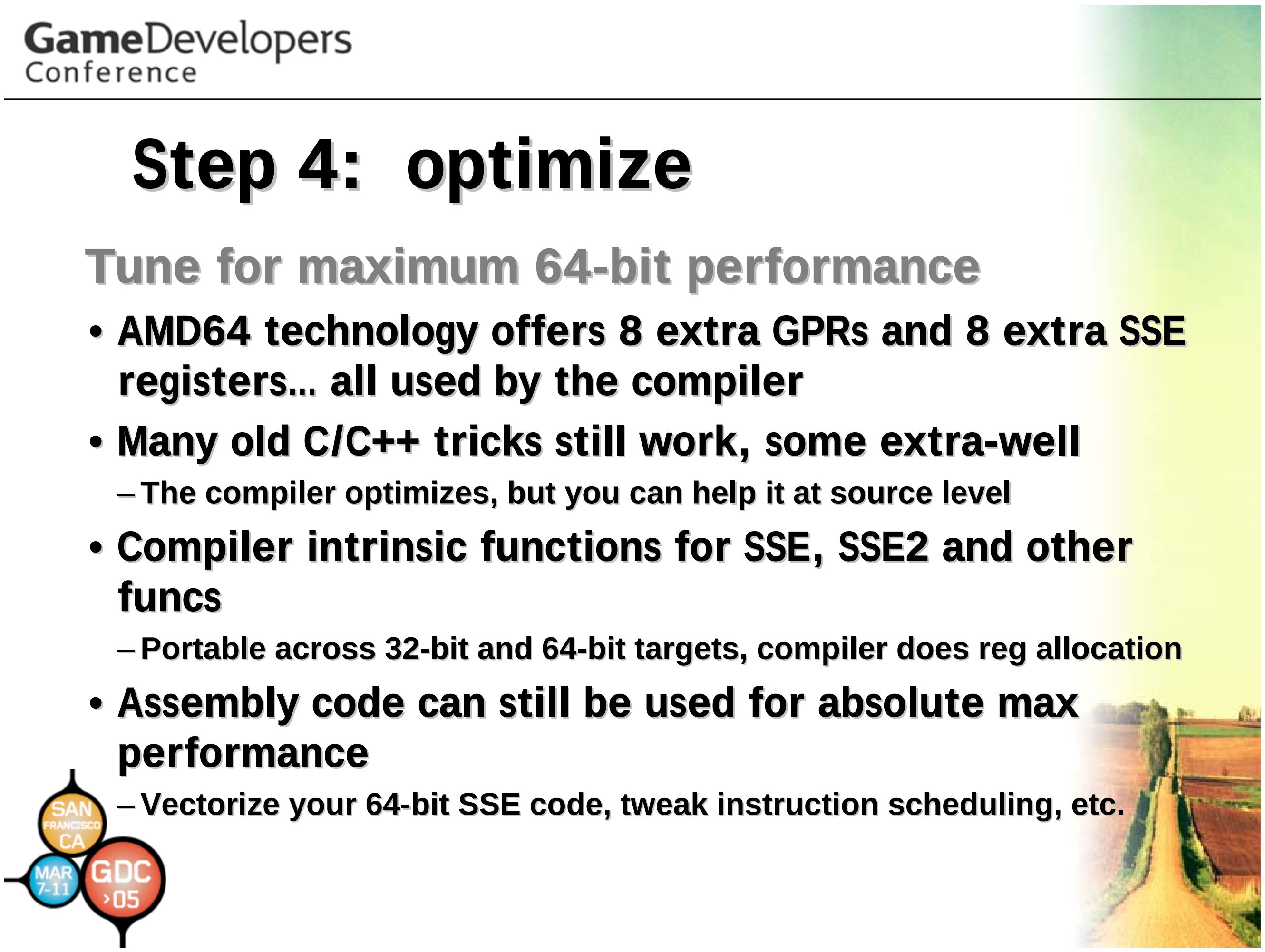
Dependent upon instructions:

Retire Cycle	Address	Disassembly	CodeBytes	Delta	Notes
3276	0x401430	movlpd xmm3,[$+00032...	66 0F 12 1D 68 2...	0	
3277	0x401438	movlpd xmm2,[$+00032...	66 0F 12 15 C0 2...	0	
3277	0x401440	lea eax,[rbp-02h]	8D 45 FE	0	
3277	0x401443	lea rdx,[rsp+50h]	48 8D 54 24 50	0	
3278	0x401448	lea rcx,[rsp+30h]	48 8D 4C 24 30	2	
3278	0x40144D	mov r8d,000003e8h	41 B8 E8 03 00 00	0	
3278	0x401453	movsd xmm0,xmm9	F2 41 0F 10 C1	5	
3315	0x401458	cvtsi2sd xmm1,ebp	F2 0F 2A CD	60	
3315	0x40145C	mulsd xmm0,xmm3	F2 0F 59 C3	0	
3320	0x401460	addsd xmm0,[$+00032...	F2 0F 58 05 70 2...	4	
3320	0x401468	mulsd xmm1,xmm3	F2 0F 59 CB	30	
3320	0x40146C	movsd [rsp+50h],xmm0	F2 0F 11 44 24 50	0	
3321	0x401472	movsd [rsp+58h],xmm0	F2 0F 11 44 24 58	1	
3321	0x401478	movsd [rsp+60h],xmm0	F2 0F 11 44 24 60	1	
3321	0x40147E	movsd [rsp+68h],xmm0	F2 0F 11 44 24 68	1	
3324	0x401484	cvtsi2sd xmm0,eax	F2 0F 2A C0	32	Bank conflict
3324	0x401488	addsd xmm1,xmm2	F2 0F 58 CA	3	
3330	0x40148C	lea eax,[rbp-01h]	8D 45 FF	0	

- **You can get a detailed pipeline view of critical code sections**

SAN FRANCISCO CA MAR 7-11 GDC >05

CodeAnalyst can be downloaded at www.amd.com developer section



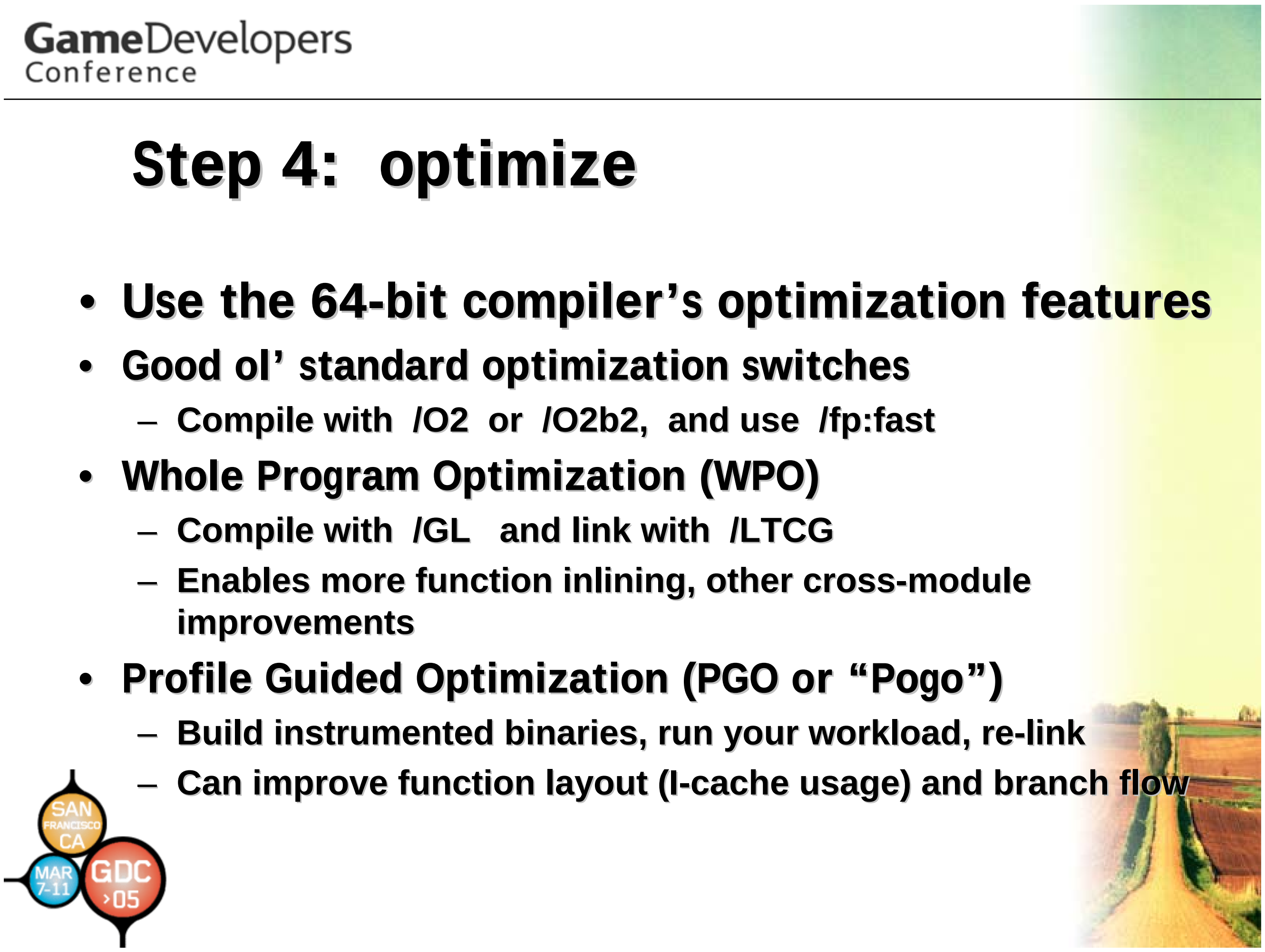
GameDevelopers Conference

Step 4: optimize

Tune for maximum 64-bit performance

- AMD64 technology offers 8 extra GPRs and 8 extra SSE registers… all used by the compiler
- Many old C/C++ tricks still work, some extra-well
  - The compiler optimizes, but you can help it at source level
- Compiler intrinsic functions for SSE, SSE2 and other funcs
  - Portable across 32-bit and 64-bit targets, compiler does reg allocation
- Assembly code can still be used for absolute max performance
  - Vectorize your 64-bit SSE code, tweak instruction scheduling, etc.

SAN FRANCISCO CA MAR 7-11 GDC ›05

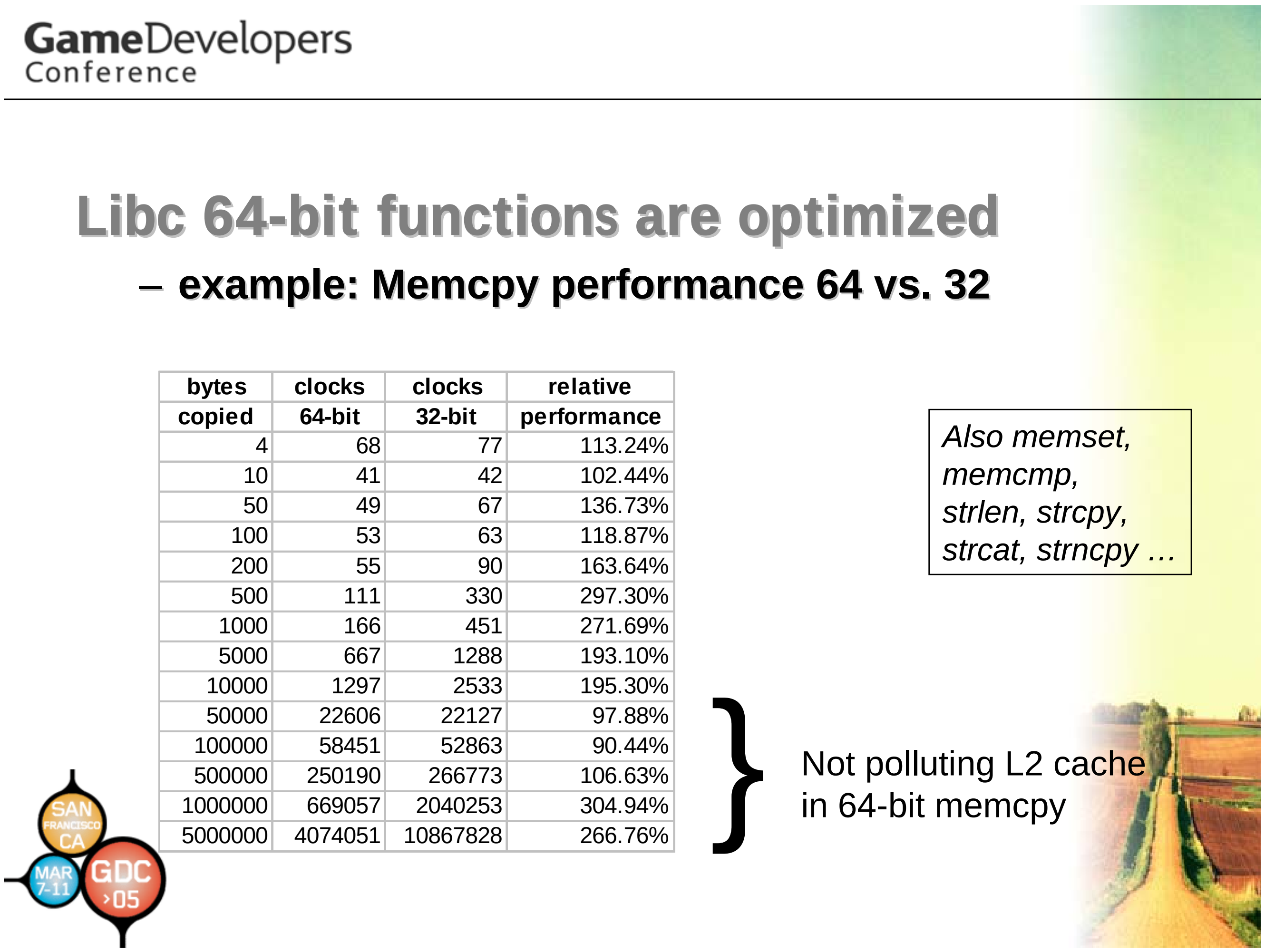


GameDevelopers Conference

Step 4: optimize

- Use the 64-bit compiler's optimization features
- Good ol' standard optimization switches
  - Compile with /O2 or /O2b2, and use /fp:fast
- Whole Program Optimization (WPO)
  - Compile with /GL and link with /LTCG
  - Enables more function inlining, other cross-module improvements
- Profile Guided Optimization (PGO or "Pogo")
  - Build instrumented binaries, run your workload, re-link
  - Can improve function layout (I-cache usage) and branch flow

SAN FRANCISCO CA MAR 7-11 GDC ›05



GameDevelopers Conference

Libc 64-bit functions are optimized

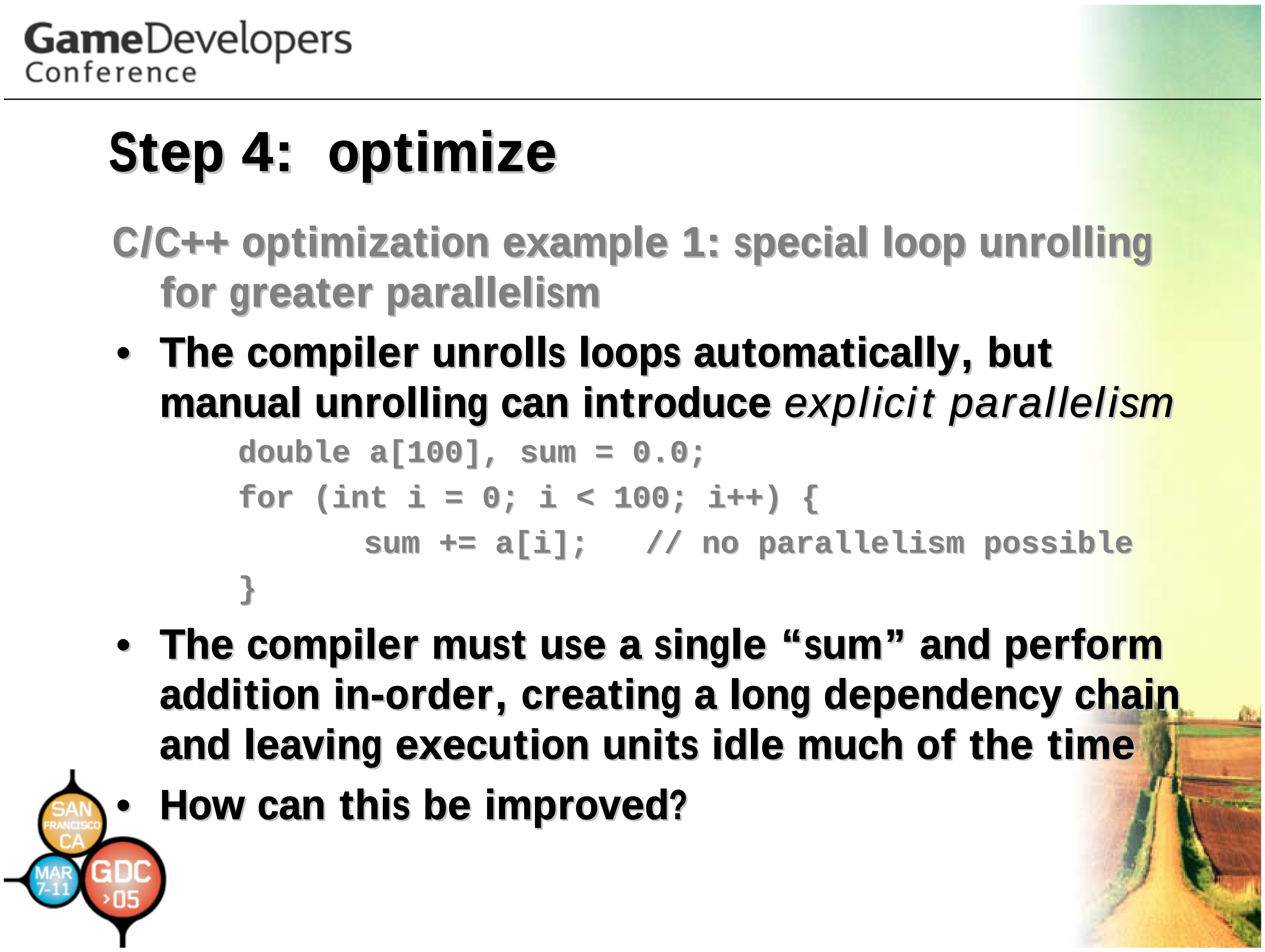
– example: Memcpy performance 64 vs. 32

bytes copied	clocks 64-bit	clocks 32-bit	relative performance
4	68	77	113.24%
10	41	42	102.44%
50	49	67	136.73%
100	53	63	118.87%
200	55	90	163.64%
500	111	330	297.30%
1000	166	451	271.69%
5000	667	1288	193.10%
10000	1297	2533	195.30%
50000	22606	22127	97.88%
100000	58451	52863	90.44%
500000	250190	266773	106.63%
1000000	669057	2040253	304.94%
5000000	4074051	10867828	266.76%

Also memset, memcmp, strlen, strcpy, strcat, strncpy …

$\}$ Not polluting L2 cache in 64-bit memcpy

SAN FRANCISCO CA MAR 7-11 GDC ›05



GameDevelopers Conference

Step 4: optimize

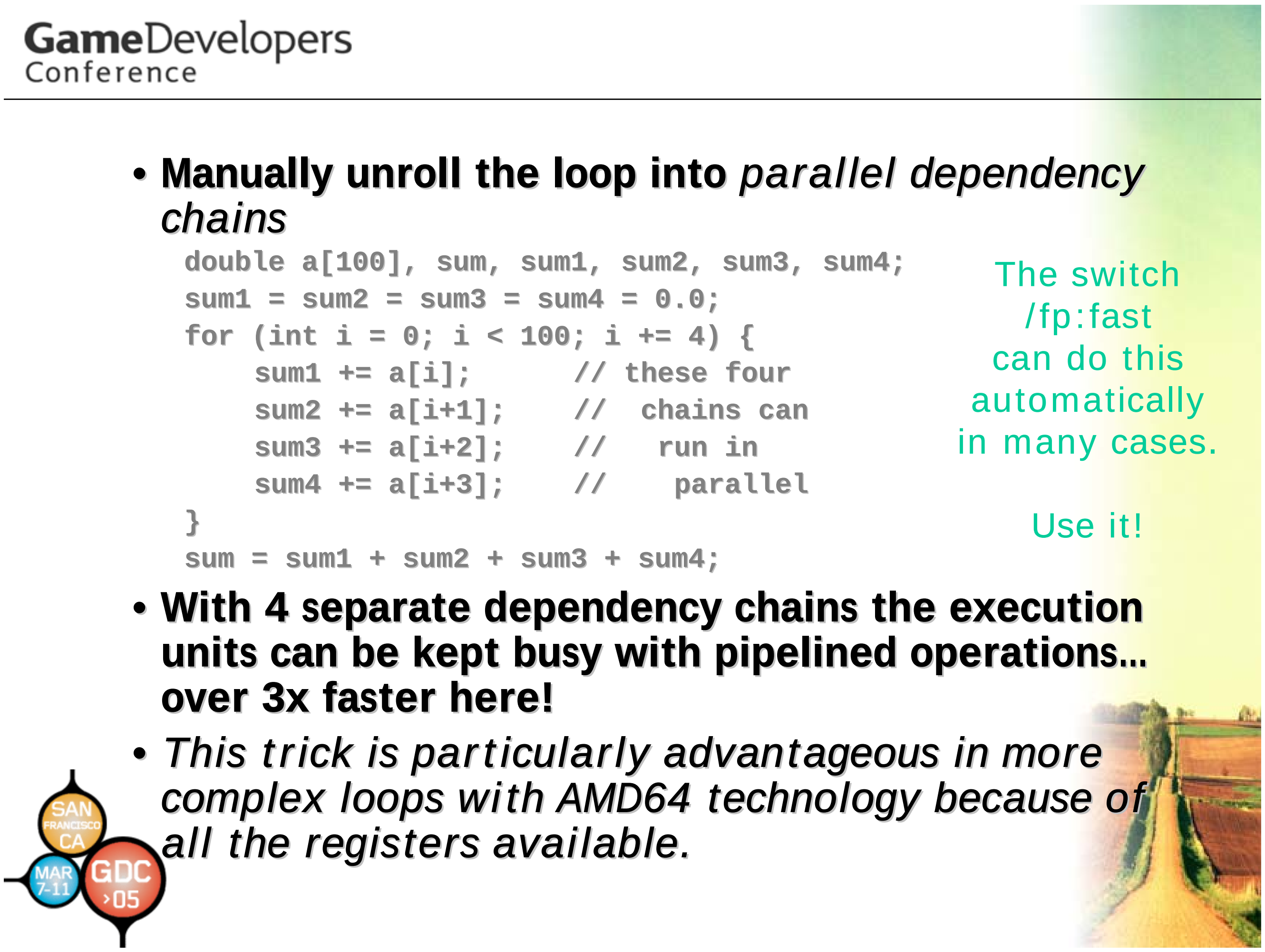
C/C++ optimization example 1: special loop unrolling for greater parallelism

- The compiler unrolls loops automatically, but manual unrolling can introduce *explicit parallelism*

```
double a[100], sum = 0.0;
for (int i = 0; i < 100; i++) {
        sum += a[i];   // no parallelism possible
}
```

- The compiler must use a single “sum” and perform addition in-order, creating a long dependency chain and leaving execution units idle much of the time
- How can this be improved?

SAN FRANCISCO CA MAR 7-11 GDC ›05



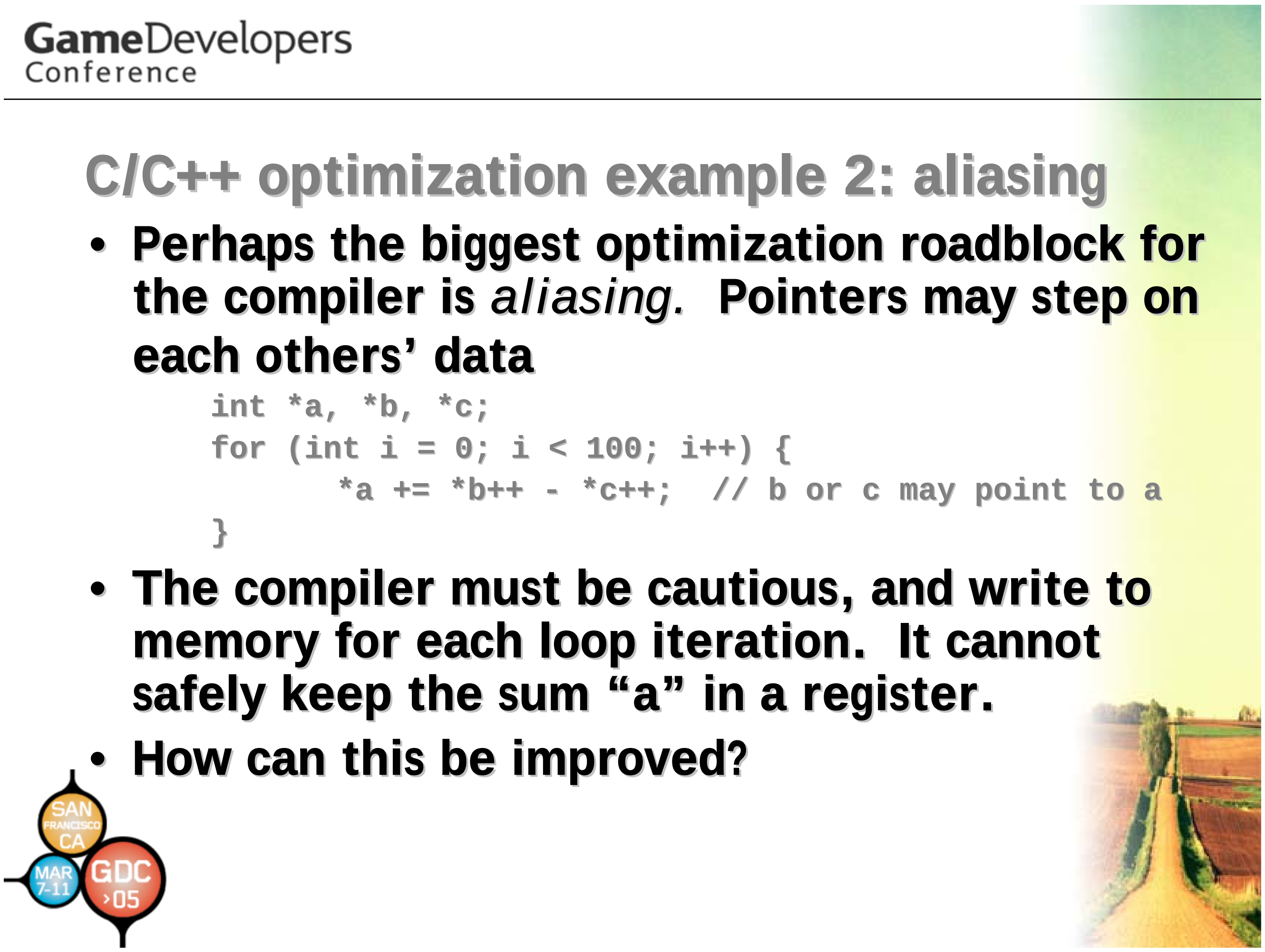
- **Manually unroll the loop into** *parallel dependency chains*

```
double a[100], sum, sum1, sum2, sum3, sum4;
sum1 = sum2 = sum3 = sum4 = 0.0;
for (int i = 0; i < 100; i += 4) {
     sum1 += a[i];       // these four
     sum2 += a[i+1];     //  chains can
     sum3 += a[i+2];     //   run in
     sum4 += a[i+3];     //    parallel
}
sum = sum1 + sum2 + sum3 + sum4;
```

The switch
/fp:fast
can do this
automatically
in many cases.

Use it!

- **With 4 separate dependency chains the execution units can be kept busy with pipelined operations… over 3x faster here!**
- *This trick is particularly advantageous in more complex loops with AMD64 technology because of all the registers available.*



GameDevelopers Conference

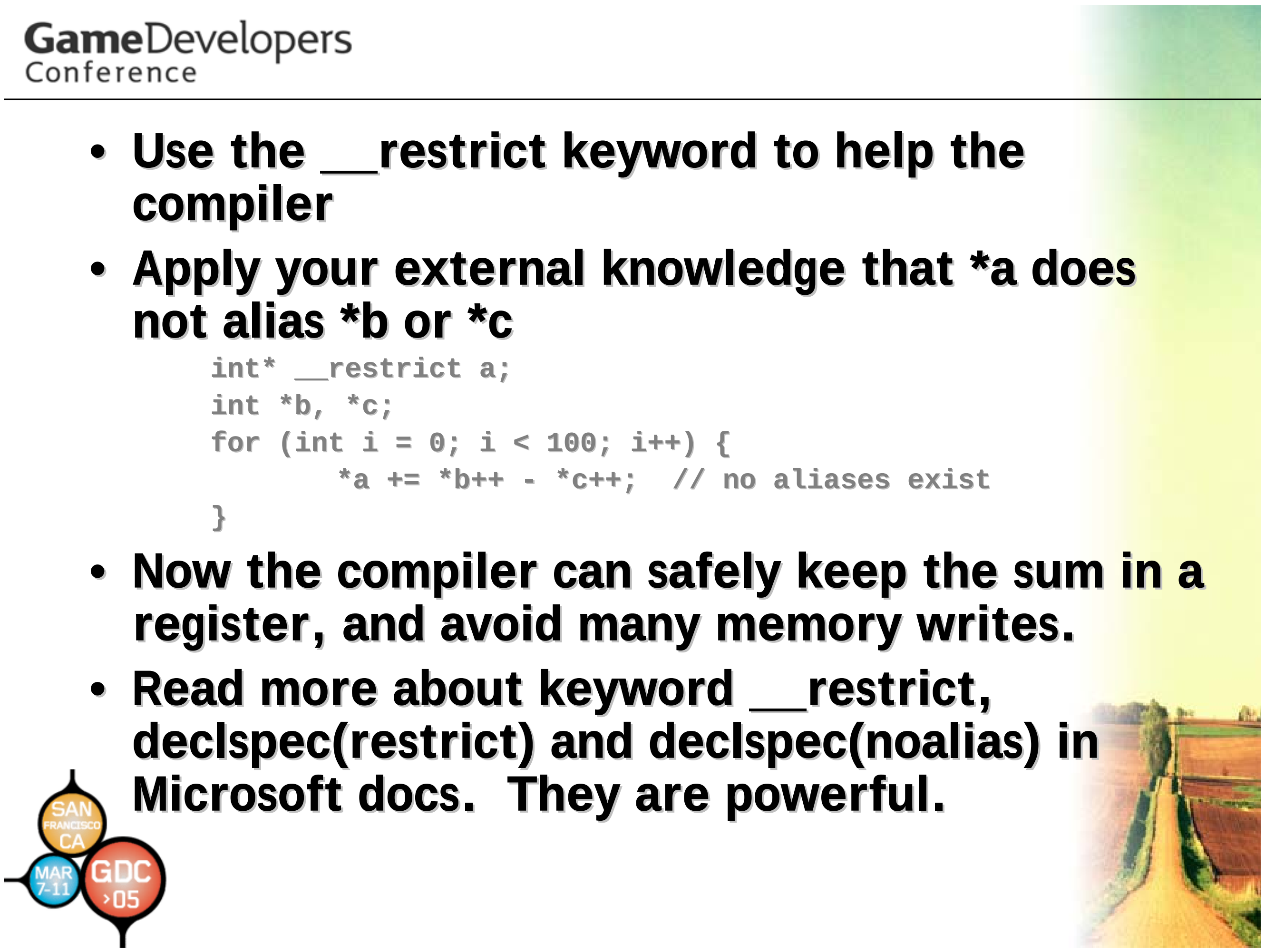
C/C++ optimization example 2: aliasing

- Perhaps the biggest optimization roadblock for the compiler is *aliasing*. Pointers may step on each others' data

```
int *a, *b, *c;
for (int i = 0; i < 100; i++) {
        *a += *b++ - *c++;  // b or c may point to a
}
```

- The compiler must be cautious, and write to memory for each loop iteration. It cannot safely keep the sum "a" in a register.
- How can this be improved?

SAN FRANCISCO CA MAR 7-11 GDC ›05

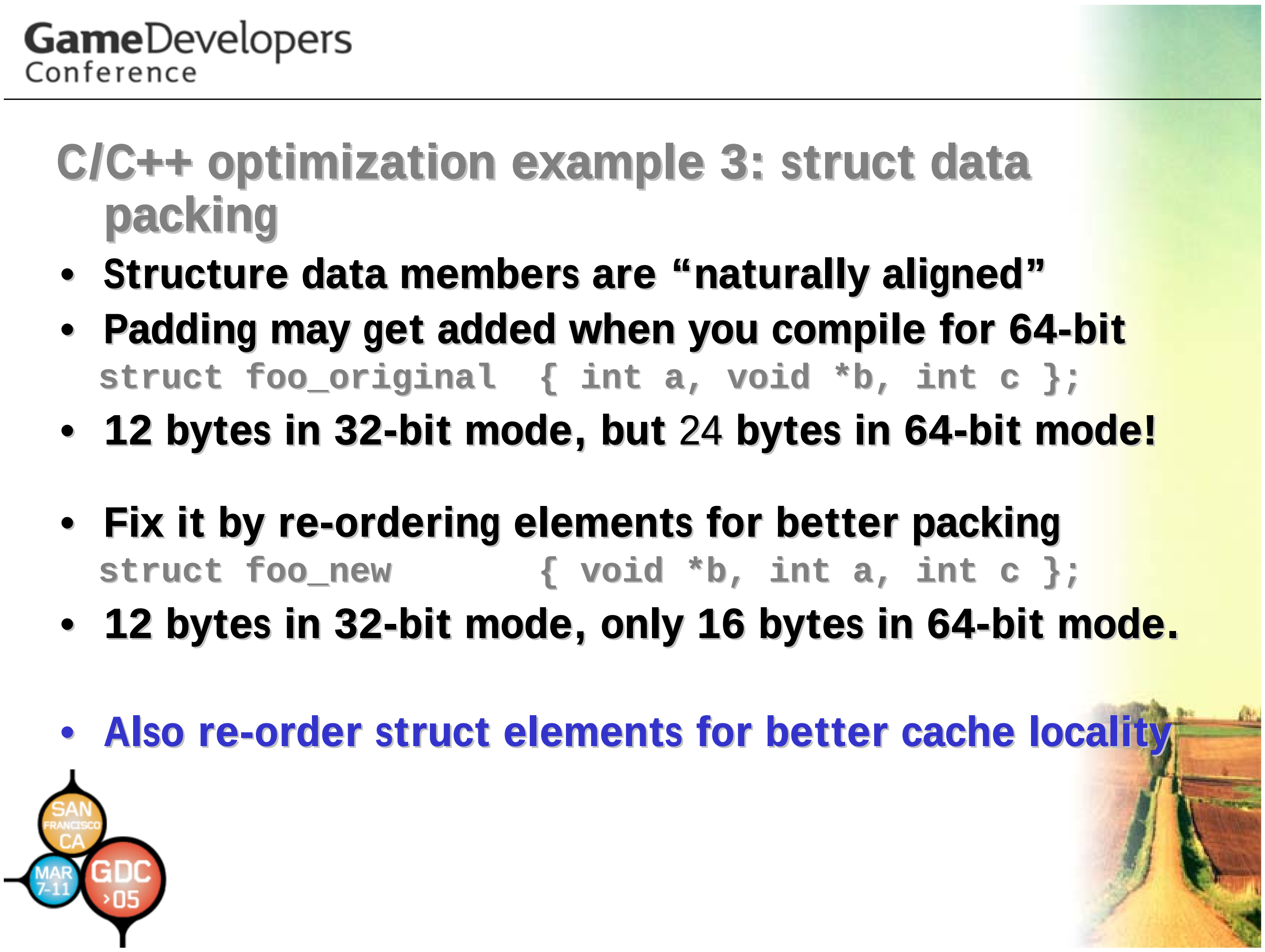


GameDevelopers Conference

- Use the __restrict keyword to help the compiler
- Apply your external knowledge that *a does not alias *b or *c

```
int* __restrict a;
int *b, *c;
for (int i = 0; i < 100; i++) {
        *a += *b++ - *c++;  // no aliases exist
}
```

- Now the compiler can safely keep the sum in a register, and avoid many memory writes.
- Read more about keyword __restrict, declspec(restrict) and declspec(noalias) in Microsoft docs. They are powerful.



GameDevelopers Conference

C/C++ optimization example 3: struct data packing

- Structure data members are “naturally aligned”
- Padding may get added when you compile for 64-bit

```
struct foo_original  { int a, void *b, int c };
```

- 12 bytes in 32-bit mode, but 24 bytes in 64-bit mode!

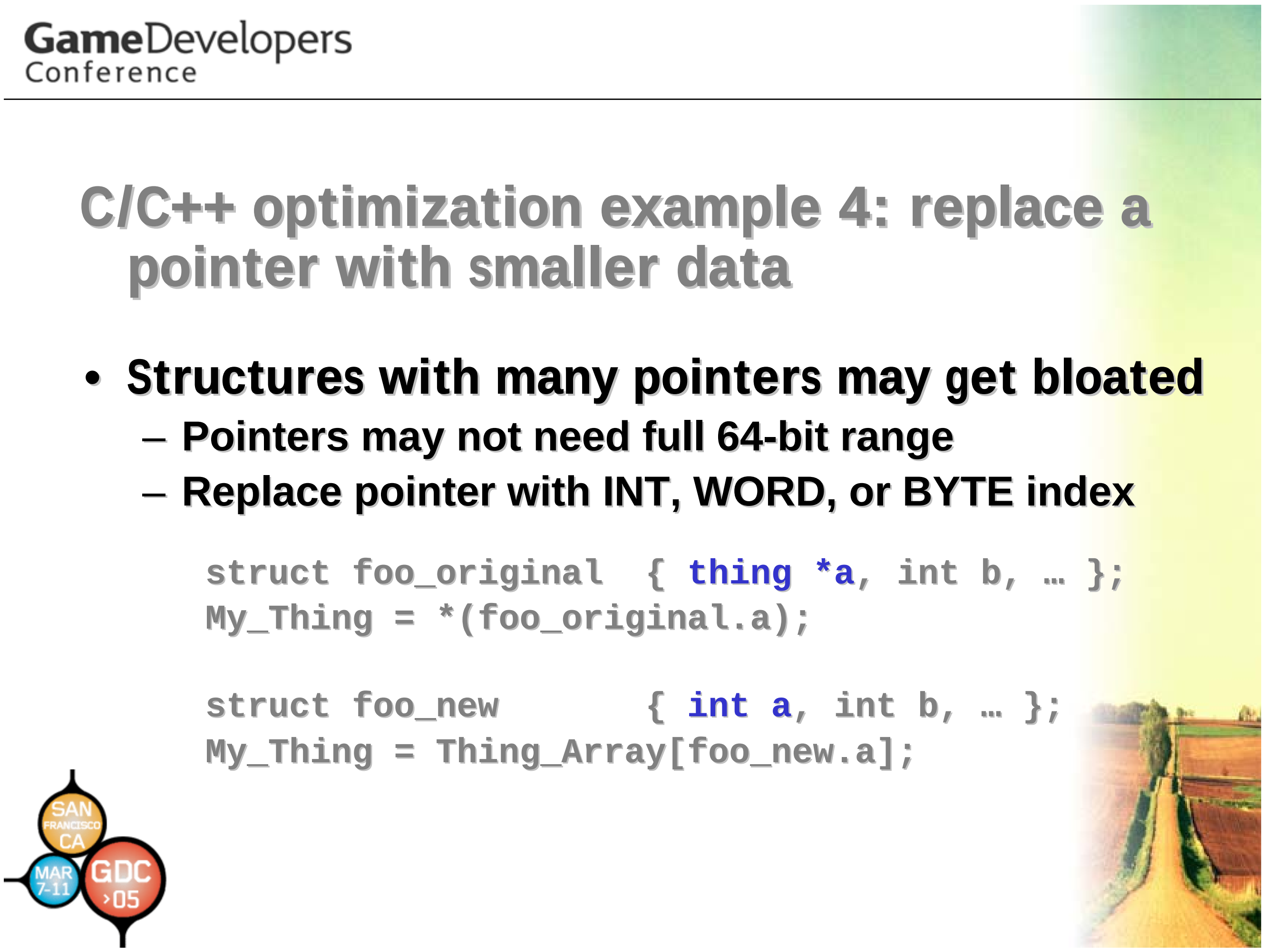
- Fix it by re-ordering elements for better packing

```
struct foo_new       { void *b, int a, int c };
```

- 12 bytes in 32-bit mode, only 16 bytes in 64-bit mode.

- Also re-order struct elements for better cache locality

SAN FRANCISCO CA MAR 7-11 GDC ›05

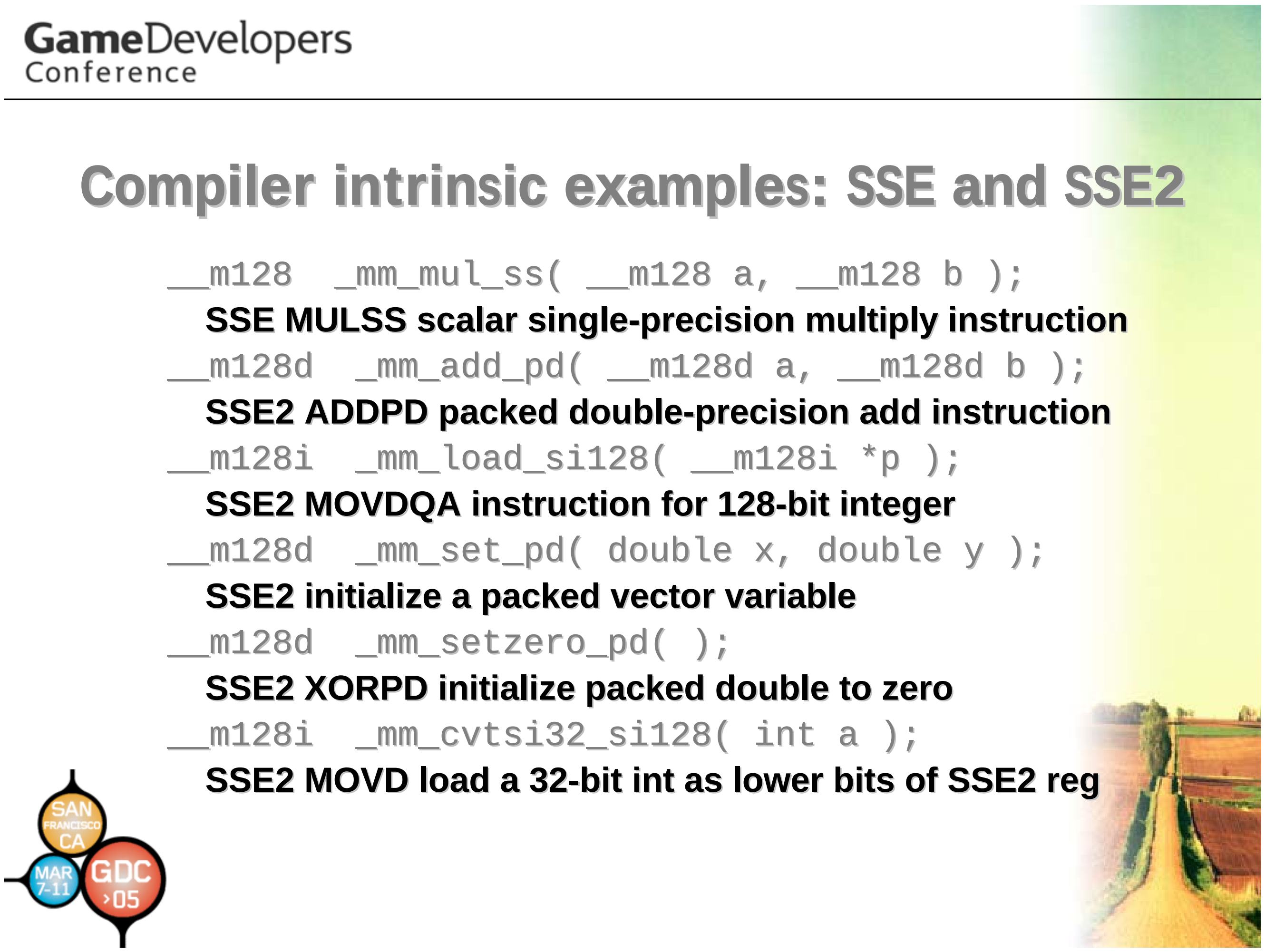


C/C++ optimization example 4: replace a pointer with smaller data

- Structures with many pointers may get bloated
  - Pointers may not need full 64-bit range
  - Replace pointer with INT, WORD, or BYTE index

```
struct foo_original  { thing *a, int b, … };
My_Thing = *(foo_original.a);

struct foo_new       { int a, int b, … };
My_Thing = Thing_Array[foo_new.a];
```

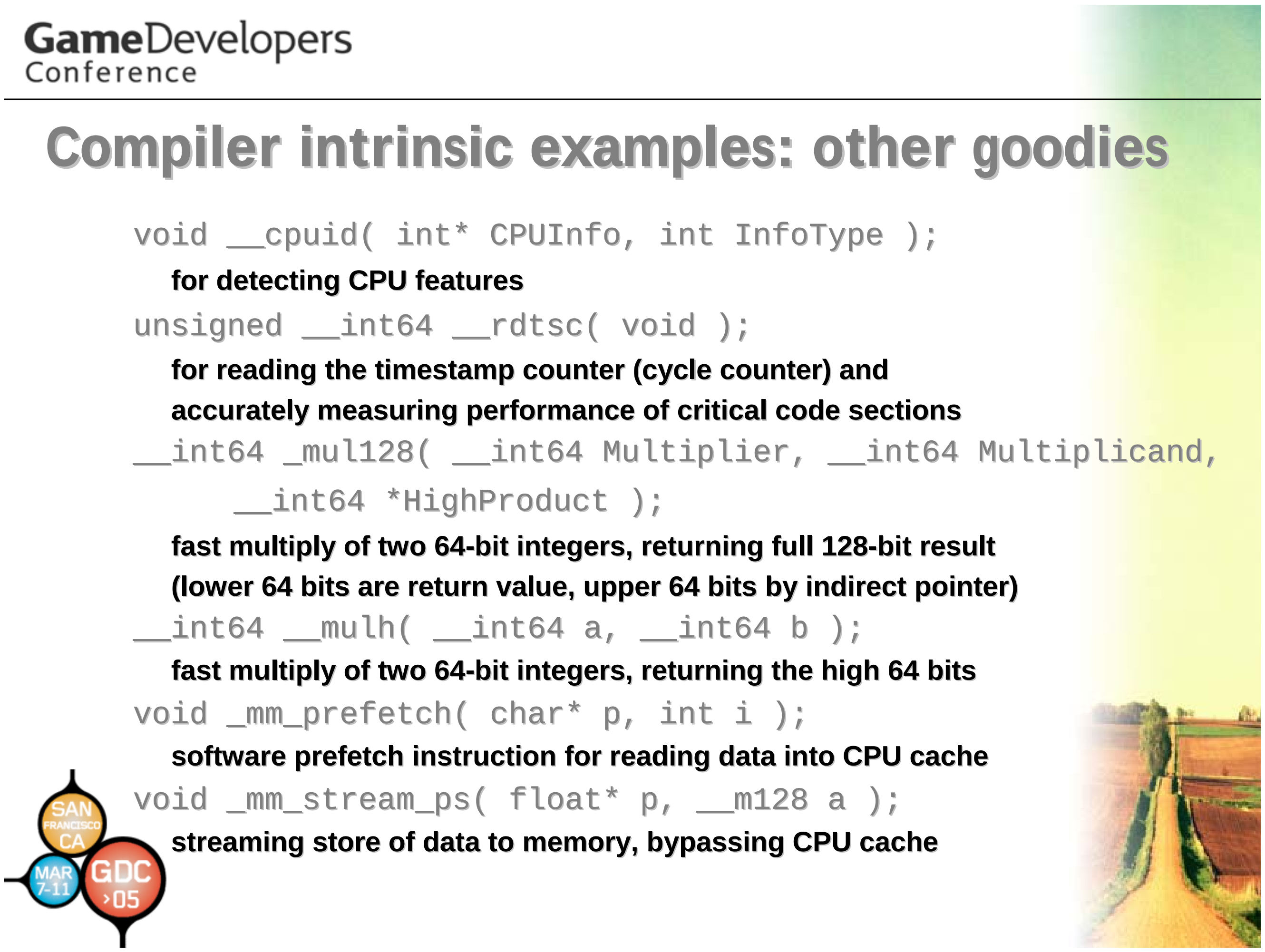


GameDevelopers Conference

Compiler intrinsic examples: SSE and SSE2

```
__m128  _mm_mul_ss( __m128 a, __m128 b );
```
SSE MULSS scalar single-precision multiply instruction
```
__m128d  _mm_add_pd( __m128d a, __m128d b );
```
SSE2 ADDPD packed double-precision add instruction
```
__m128i  _mm_load_si128( __m128i *p );
```
SSE2 MOVDQA instruction for 128-bit integer
```
__m128d  _mm_set_pd( double x, double y );
```
SSE2 initialize a packed vector variable
```
__m128d  _mm_setzero_pd( );
```
SSE2 XORPD initialize packed double to zero
```
__m128i  _mm_cvtsi32_si128( int a );
```
SSE2 MOVD load a 32-bit int as lower bits of SSE2 reg

SAN FRANCISCO CA MAR 7-11 GDC ›05



GameDevelopers Conference

Compiler intrinsic examples: other goodies

```
void __cpuid( int* CPUInfo, int InfoType );
```

for detecting CPU features

```
unsigned __int64 __rdtsc( void );
```

for reading the timestamp counter (cycle counter) and accurately measuring performance of critical code sections

```
__int64 _mul128( __int64 Multiplier, __int64 Multiplicand,
     __int64 *HighProduct );
```

fast multiply of two 64-bit integers, returning full 128-bit result (lower 64 bits are return value, upper 64 bits by indirect pointer)

```
__int64 __mulh( __int64 a, __int64 b );
```

fast multiply of two 64-bit integers, returning the high 64 bits

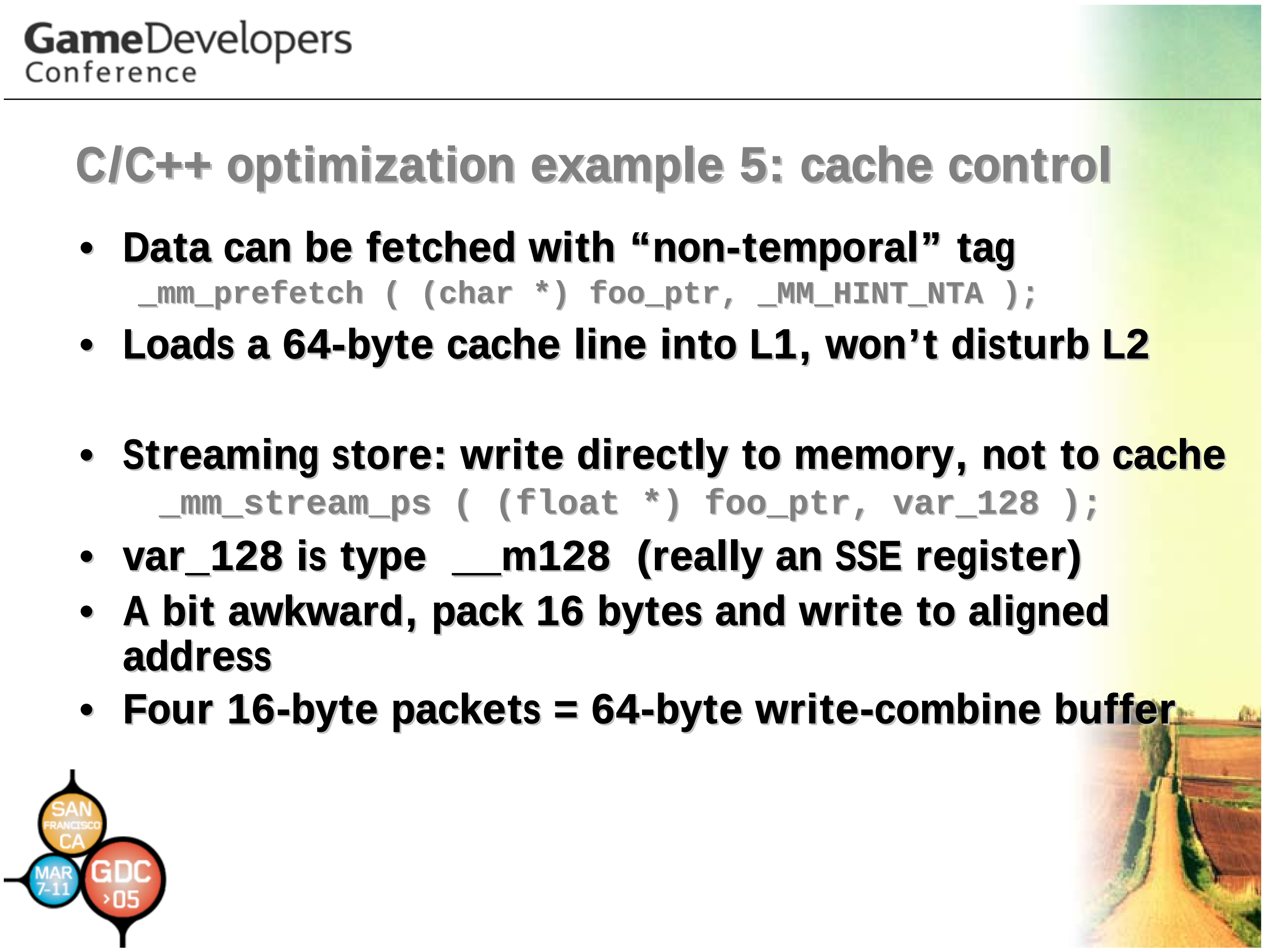
```
void _mm_prefetch( char* p, int i );
```

software prefetch instruction for reading data into CPU cache

```
void _mm_stream_ps( float* p, __m128 a );
```

streaming store of data to memory, bypassing CPU cache

SAN FRANCISCO CA MAR 7-11 GDC ›05



GameDevelopers Conference

C/C++ optimization example 5: cache control

- Data can be fetched with “non-temporal” tag

```
_mm_prefetch ( (char *) foo_ptr, _MM_HINT_NTA );
```

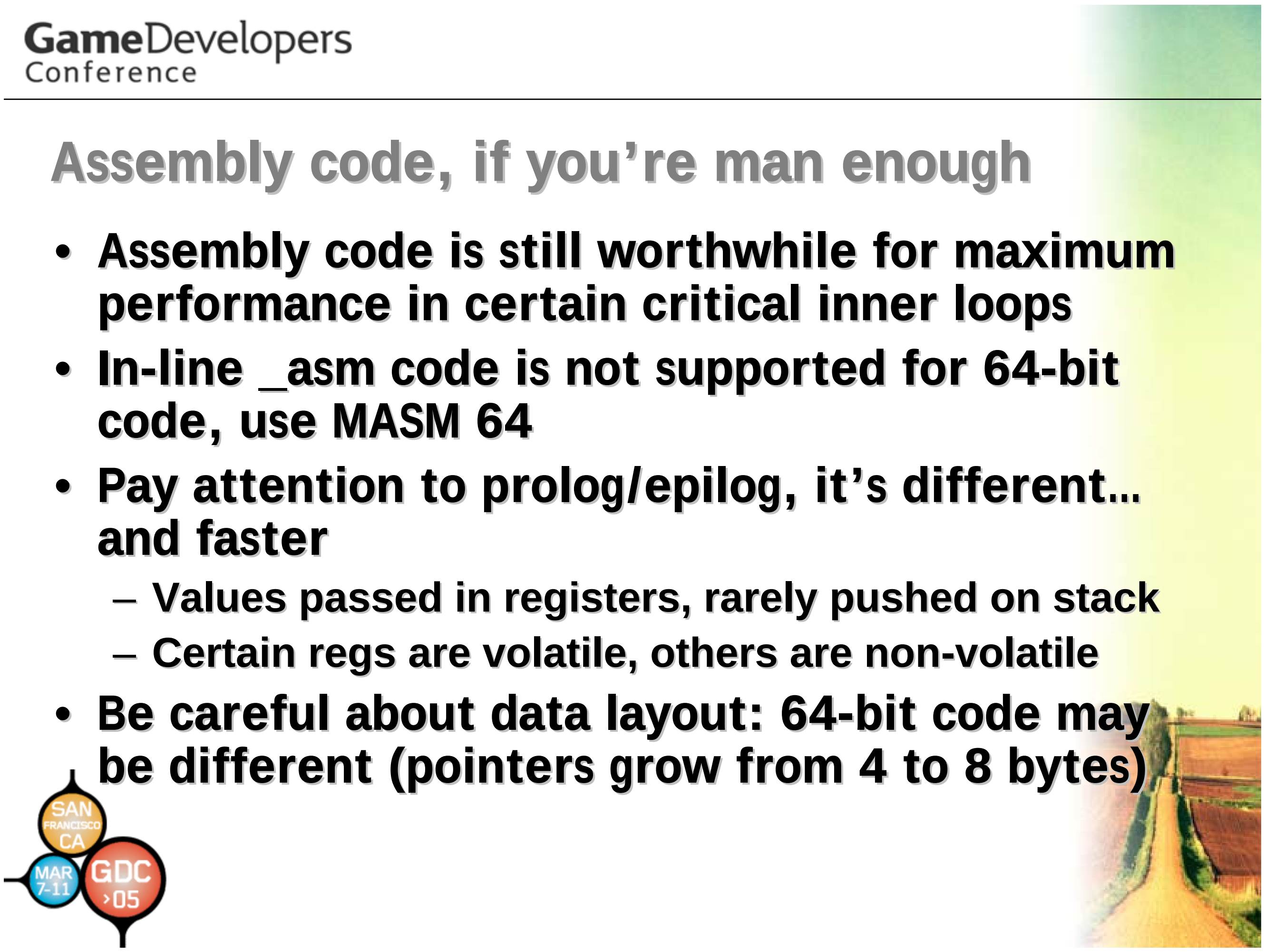
- Loads a 64-byte cache line into L1, won’t disturb L2

- Streaming store: write directly to memory, not to cache

```
_mm_stream_ps ( (float *) foo_ptr, var_128 );
```

- var_128 is type __m128 (really an SSE register)
- A bit awkward, pack 16 bytes and write to aligned address
- Four 16-byte packets = 64-byte write-combine buffer

SAN FRANCISCO CA MAR 7-11 GDC ›05



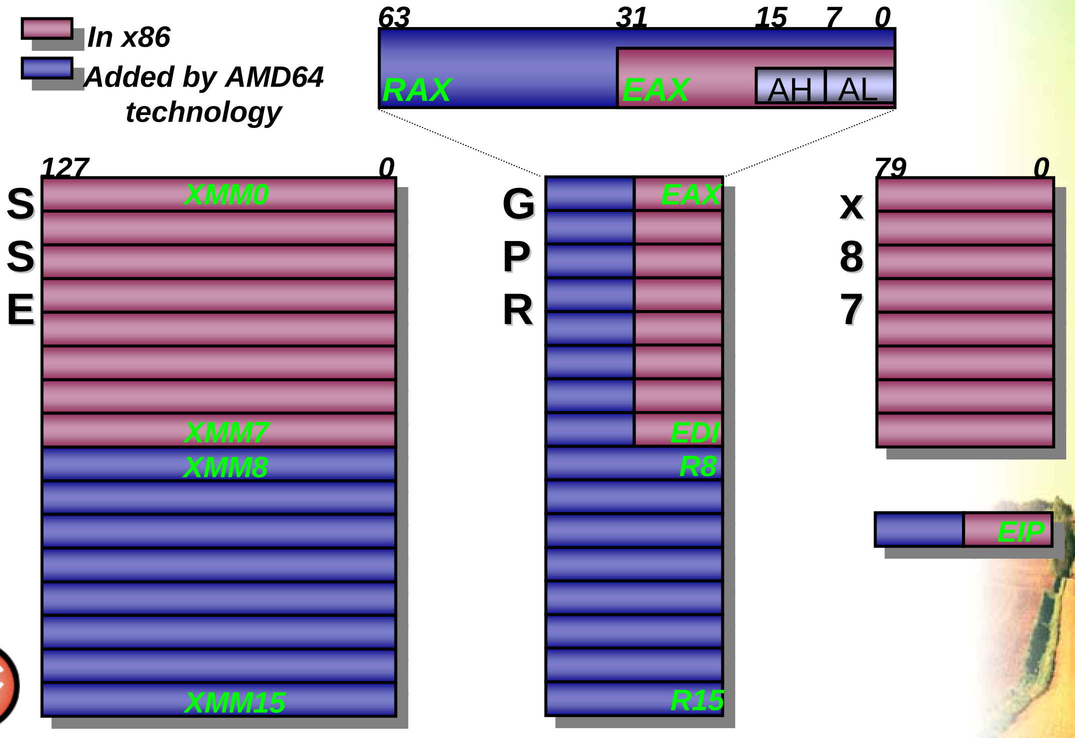
GameDevelopers Conference

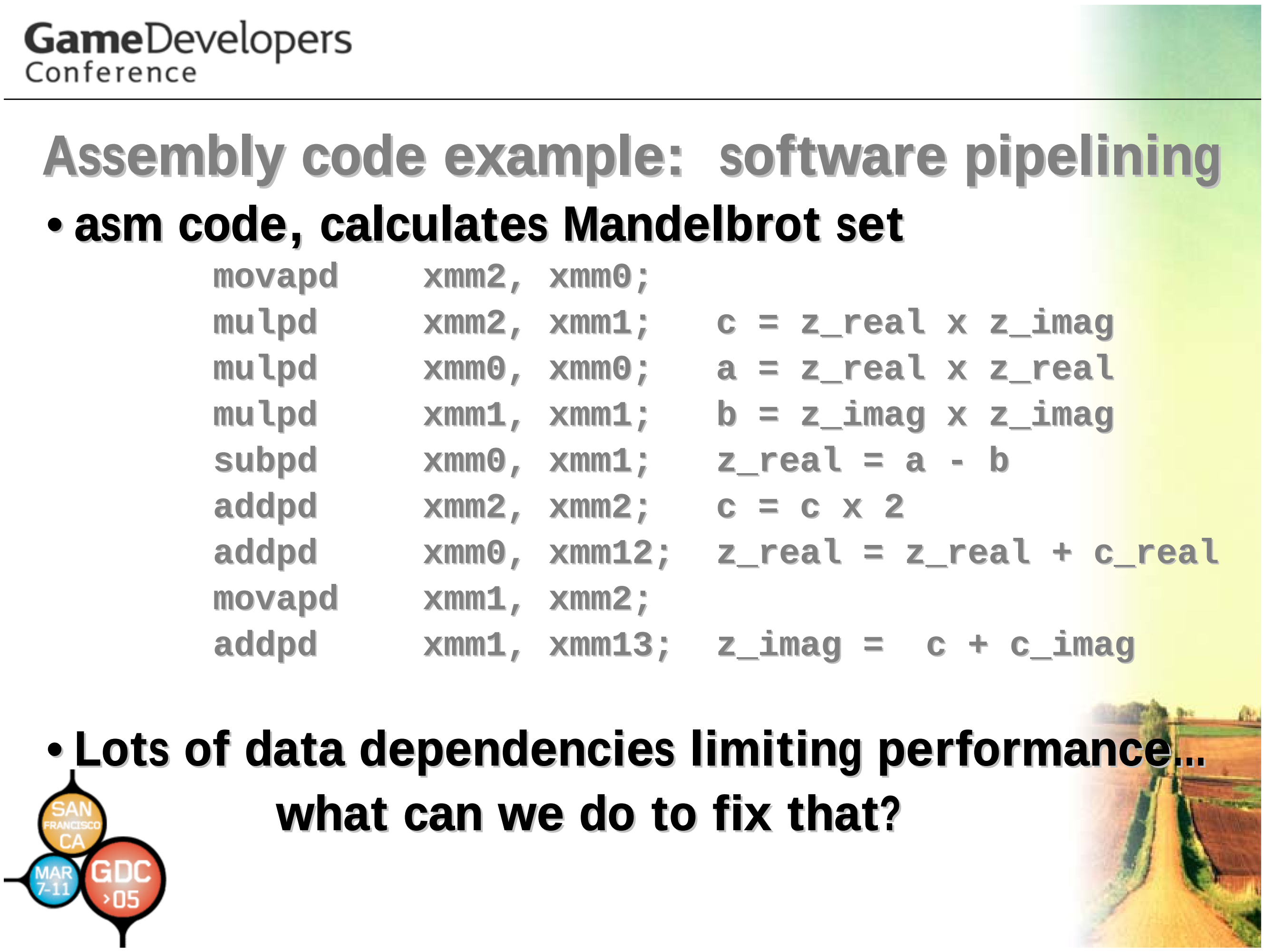
Assembly code, if you’re man enough

- Assembly code is still worthwhile for maximum performance in certain critical inner loops
- In-line _asm code is not supported for 64-bit code, use MASM 64
- Pay attention to prolog/epilog, it’s different… and faster
  - Values passed in registers, rarely pushed on stack
  - Certain regs are volatile, others are non-volatile
- Be careful about data layout: 64-bit code may be different (pointers grow from 4 to 8 bytes)

GameDevelopers Conference

AMD64 (x64) Assembly Programmer's Model





GameDevelopers Conference

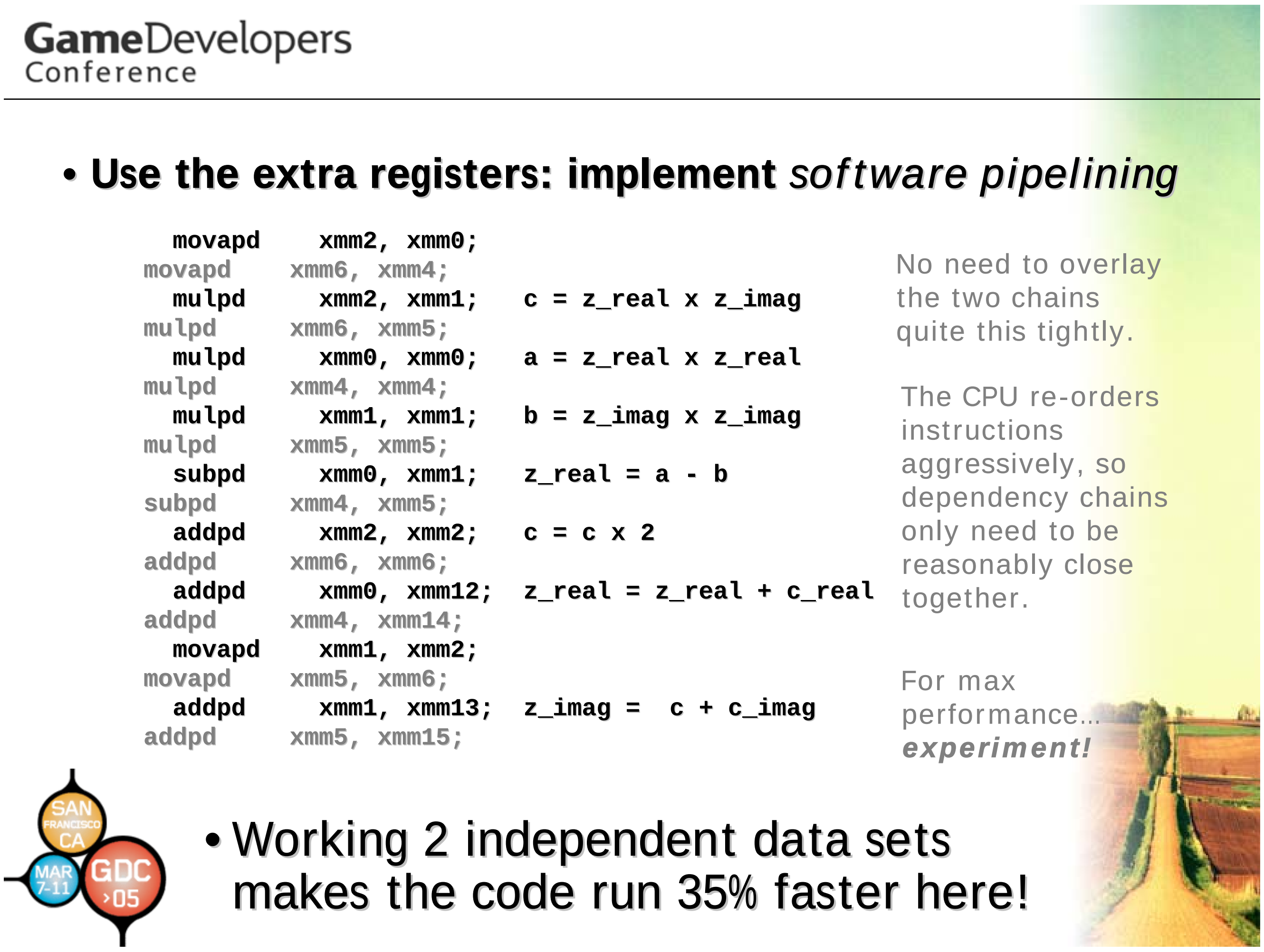
Assembly code example: software pipelining

- asm code, calculates Mandelbrot set

```
movapd     xmm2, xmm0;
mulpd      xmm2, xmm1;   c = z_real x z_imag
mulpd      xmm0, xmm0;   a = z_real x z_real
mulpd      xmm1, xmm1;   b = z_imag x z_imag
subpd      xmm0, xmm1;   z_real = a - b
addpd      xmm2, xmm2;   c = c x 2
addpd      xmm0, xmm12;  z_real = z_real + c_real
movapd     xmm1, xmm2;
addpd      xmm1, xmm13;  z_imag =  c + c_imag
```

- Lots of data dependencies limiting performance… what can we do to fix that?

SAN FRANCISCO CA MAR 7-11 GDC ›05



GameDevelopers Conference

• Use the extra registers: implement *software pipelining*

```
  movapd    xmm2, xmm0;
movapd    xmm6, xmm4;
  mulpd     xmm2, xmm1;   c = z_real x z_imag
mulpd     xmm6, xmm5;
  mulpd     xmm0, xmm0;   a = z_real x z_real
mulpd     xmm4, xmm4;
  mulpd     xmm1, xmm1;   b = z_imag x z_imag
mulpd     xmm5, xmm5;
  subpd     xmm0, xmm1;   z_real = a - b
subpd     xmm4, xmm5;
  addpd     xmm2, xmm2;   c = c x 2
addpd     xmm6, xmm6;
  addpd     xmm0, xmm12;  z_real = z_real + c_real
addpd     xmm4, xmm14;
  movapd    xmm1, xmm2;
movapd    xmm5, xmm6;
  addpd     xmm1, xmm13;  z_imag =  c + c_imag
addpd     xmm5, xmm15;
```

No need to overlay the two chains quite this tightly.

The CPU re-orders instructions aggressively, so dependency chains only need to be reasonably close together.

For max performance… ***experiment!***

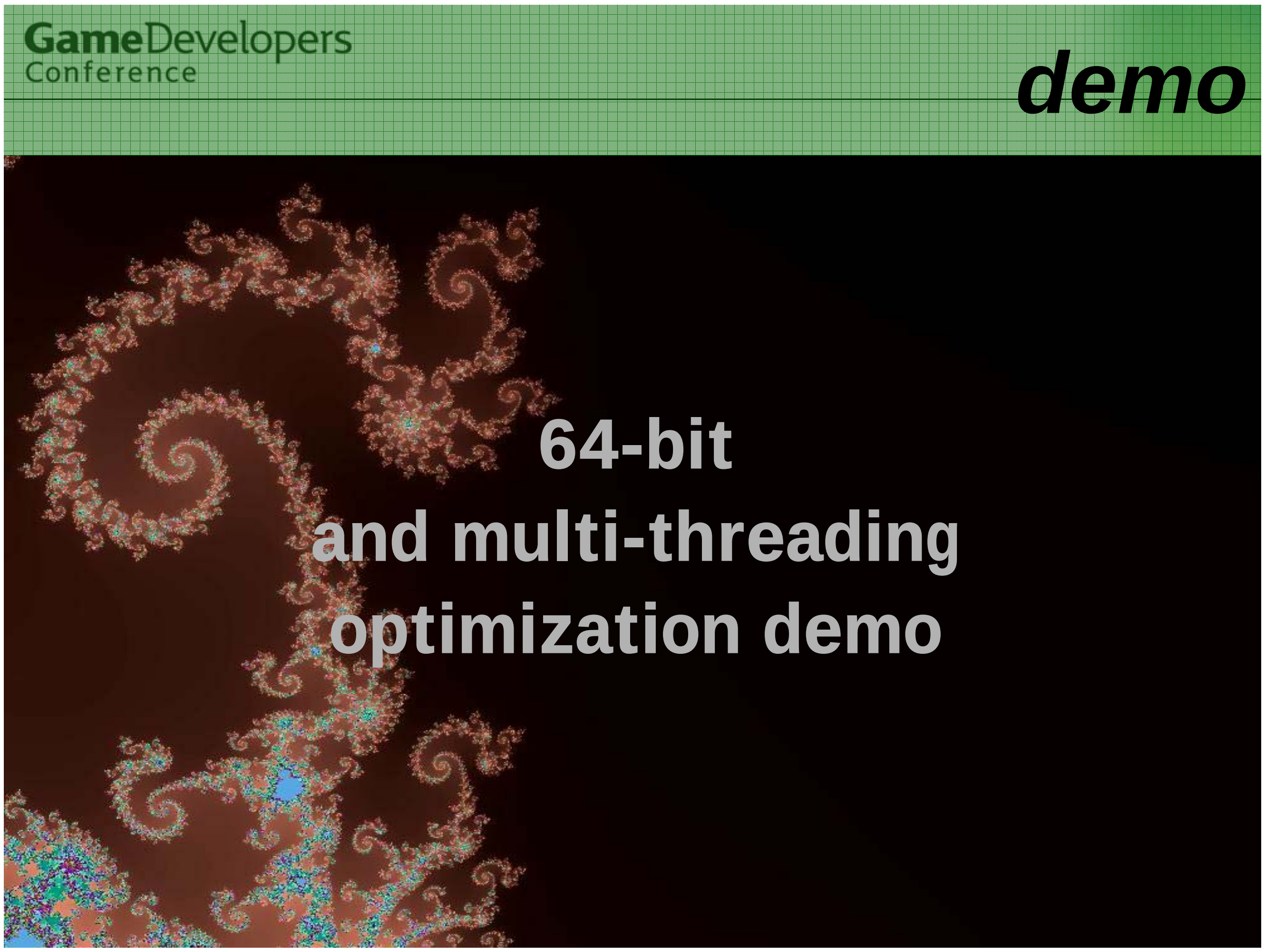
• Working 2 independent data sets makes the code run 35% faster here!

SAN FRANCISCO CA MAR 7-11 GDC ›05

GameDevelopers Conference

Step 5: beverage

- After you have your 64-bit code running like blazes…
- Enjoy a beverage of your choice
- Don’t skip this step



GameDevelopers Conference

demo

64-bit and multi-threading optimization demo

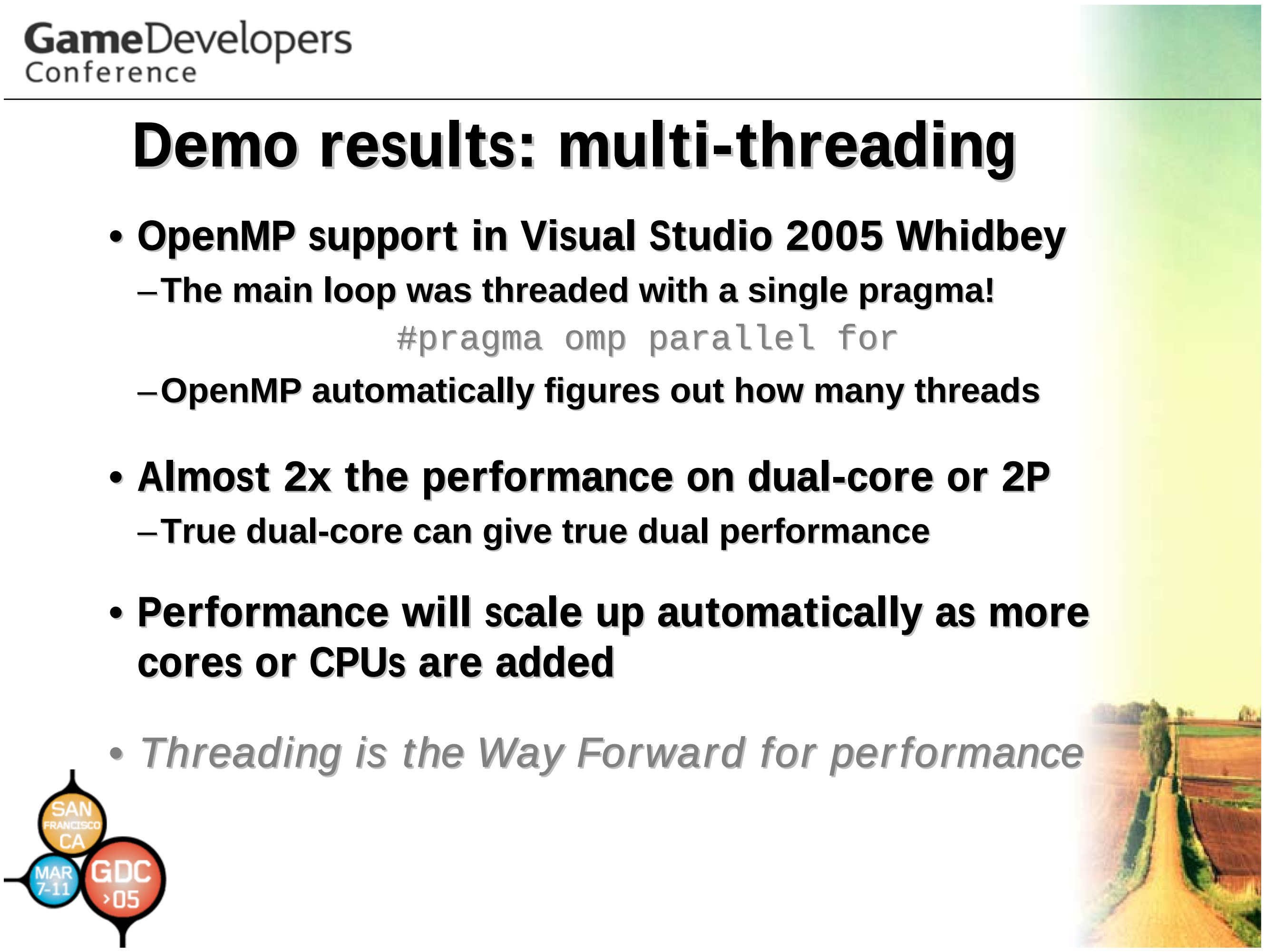
GameDevelopers Conference

Demo results: dependency chains

- Inserting a second dependency chain in the loop dramatically improves performance on both 32-bit and 64-bit
- 64-bit code benefits more, because of the extra SSE regs
- 64-bit mode provides a 30%+ boost over 32-bit mode

	32-bit	64-bit	
1 chain	.92 Gflop	1.01 Gflop	~10% 64-bit benefit
2 chains	1.31 Gflop	1.72 Gflop	~30% 64-bit benefit
Chain gain =	42%	70%	

SAN FRANCISCO CA MAR 7-11 GDC ›05

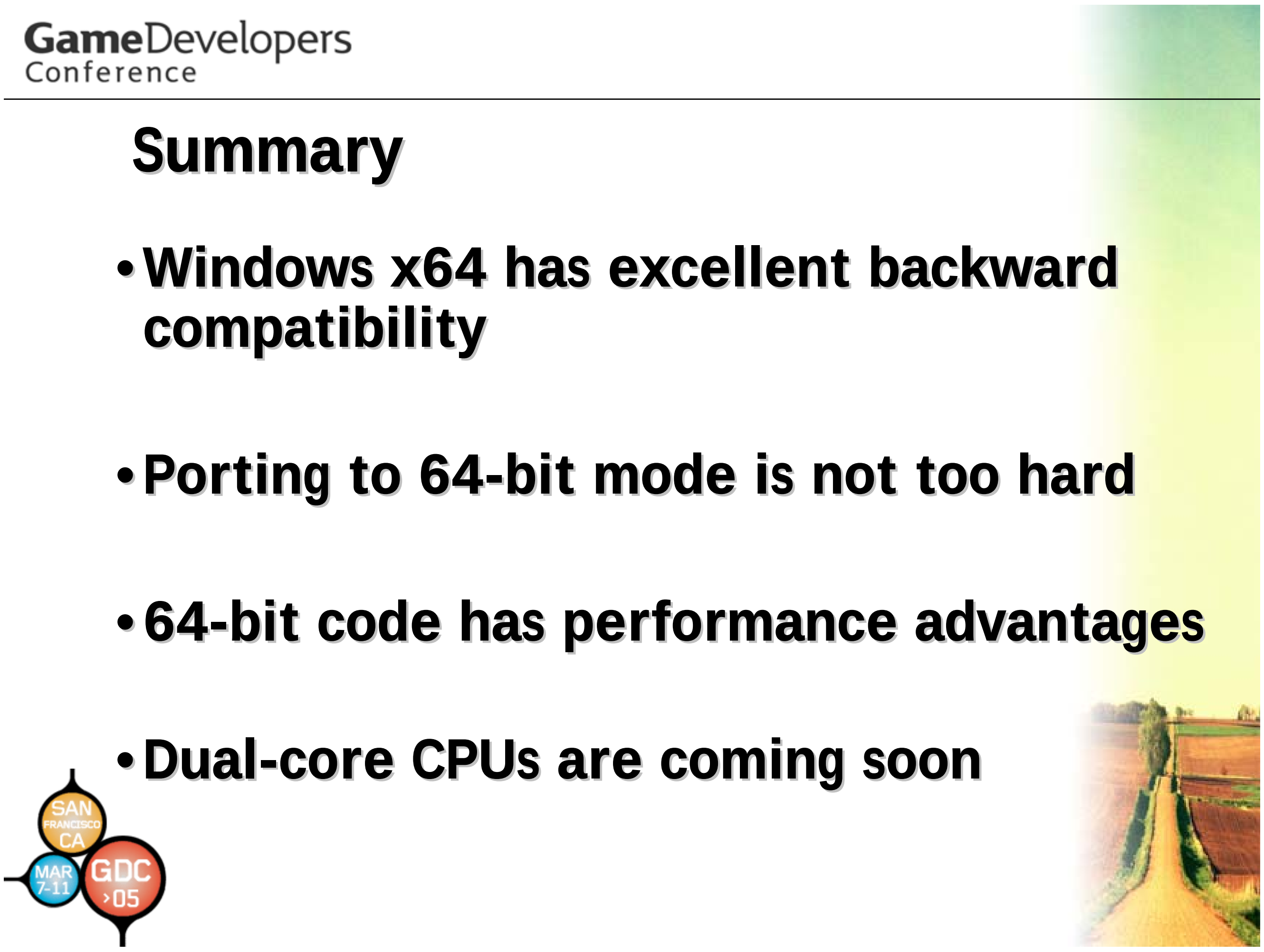


GameDevelopers Conference

Demo results: multi-threading

- OpenMP support in Visual Studio 2005 Whidbey
  - The main loop was threaded with a single pragma!

    ```
    #pragma omp parallel for
    ```
  - OpenMP automatically figures out how many threads
- Almost 2x the performance on dual-core or 2P
  - True dual-core can give true dual performance
- Performance will scale up automatically as more cores or CPUs are added
- *Threading is the Way Forward for performance*



GameDevelopers Conference

Summary

- Windows x64 has excellent backward compatibility
- Porting to 64-bit mode is not too hard
- 64-bit code has performance advantages
- Dual-core CPUs are coming soon

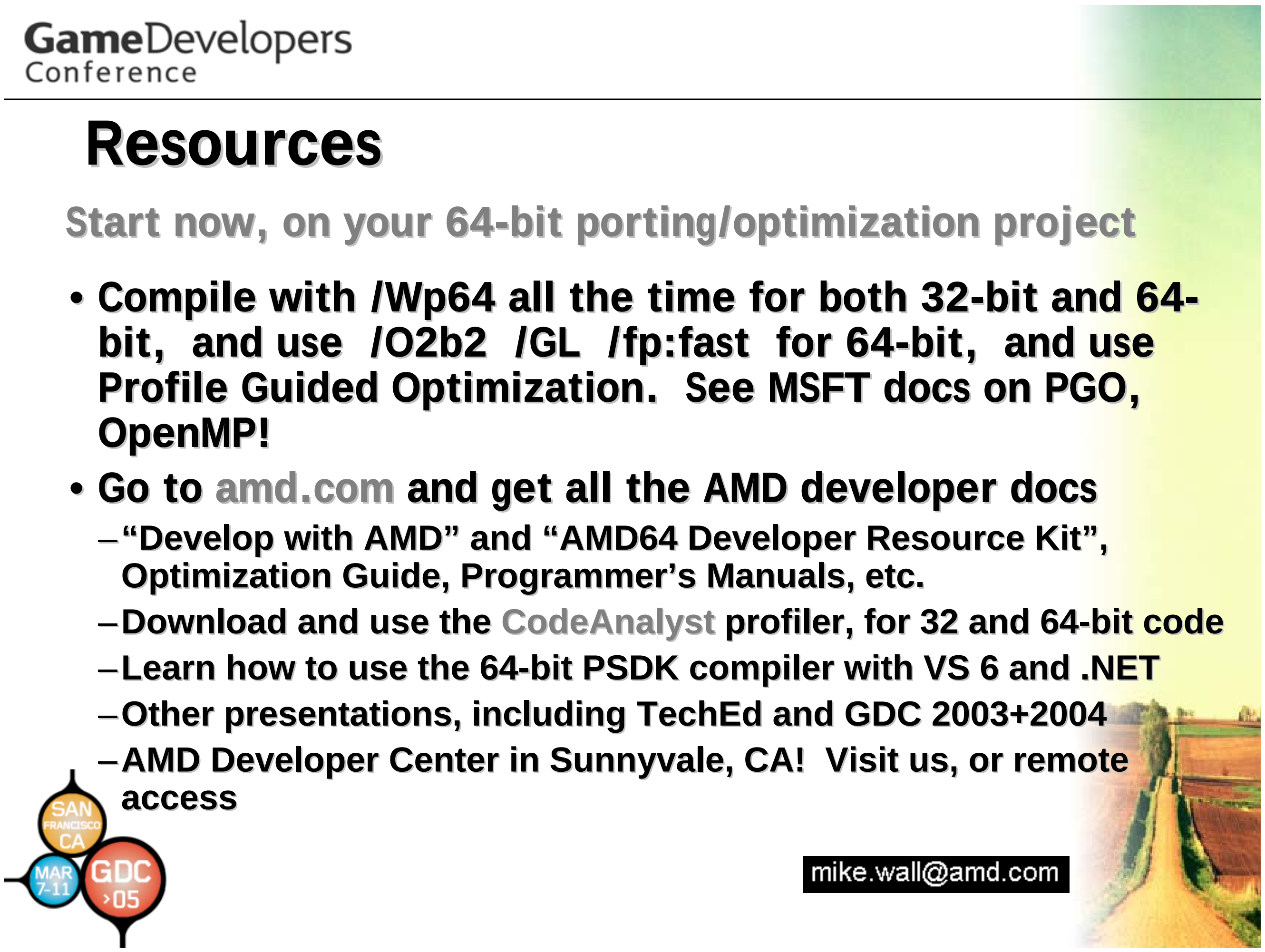
SAN FRANCISCO CA
MAR 7-11
GDC ›05

GameDevelopers Conference

Call to Action

- Test all your 32-bit games on Windows x64
  - 99% will just work fine
  - 1% need tweaks (replace 16-bit installer, tweak OS version detection)
  - All drivers need to be ported
- Port to 64-bit mode and optimize performance
  - Familiar tools, classic optimization techniques, plus a few new ones
- Multi-thread your game to leverage dual-core!
  - OpenMP makes multi-threading very straightforward
  - Relevant in *every application segment*

SAN FRANCISCO CA MAR 7-11 GDC ›05



GameDevelopers Conference

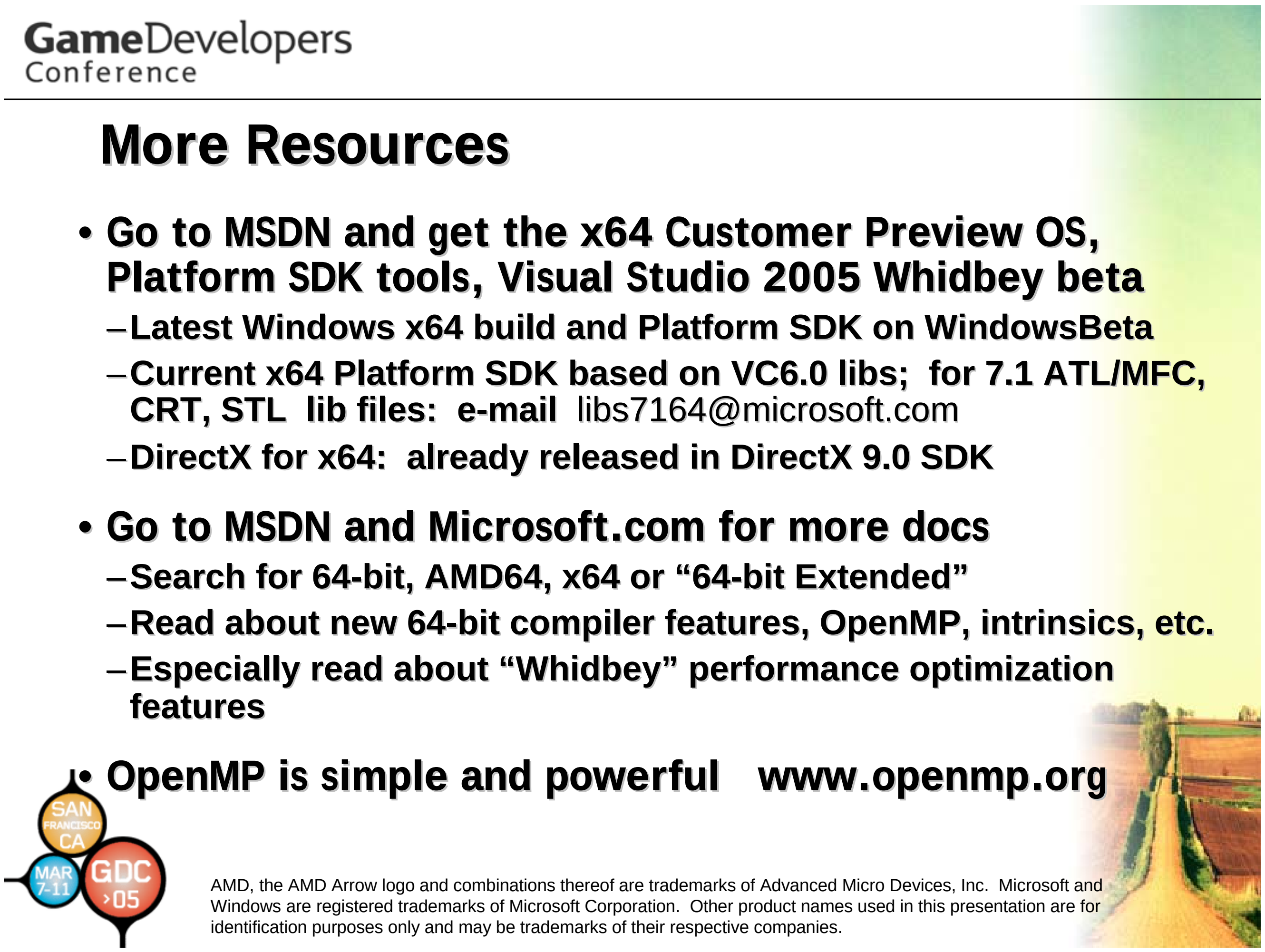
Resources

Start now, on your 64-bit porting/optimization project

- Compile with /Wp64 all the time for both 32-bit and 64-bit, and use /O2b2 /GL /fp:fast for 64-bit, and use Profile Guided Optimization. See MSFT docs on PGO, OpenMP!
- Go to amd.com and get all the AMD developer docs
  - “Develop with AMD” and “AMD64 Developer Resource Kit”, Optimization Guide, Programmer’s Manuals, etc.
  - Download and use the CodeAnalyst profiler, for 32 and 64-bit code
  - Learn how to use the 64-bit PSDK compiler with VS 6 and .NET
  - Other presentations, including TechEd and GDC 2003+2004
  - AMD Developer Center in Sunnyvale, CA! Visit us, or remote access

SAN FRANCISCO CA MAR 7-11 GDC ›05

mike.wall@amd.com



GameDevelopers Conference

More Resources

- Go to MSDN and get the x64 Customer Preview OS, Platform SDK tools, Visual Studio 2005 Whidbey beta
  - Latest Windows x64 build and Platform SDK on WindowsBeta
  - Current x64 Platform SDK based on VC6.0 libs; for 7.1 ATL/MFC, CRT, STL lib files: e-mail libs7164@microsoft.com
  - DirectX for x64: already released in DirectX 9.0 SDK
- Go to MSDN and Microsoft.com for more docs
  - Search for 64-bit, AMD64, x64 or “64-bit Extended”
  - Read about new 64-bit compiler features, OpenMP, intrinsics, etc.
  - Especially read about “Whidbey” performance optimization features
- OpenMP is simple and powerful www.openmp.org

SAN FRANCISCO CA MAR 7-11 GDC ›05