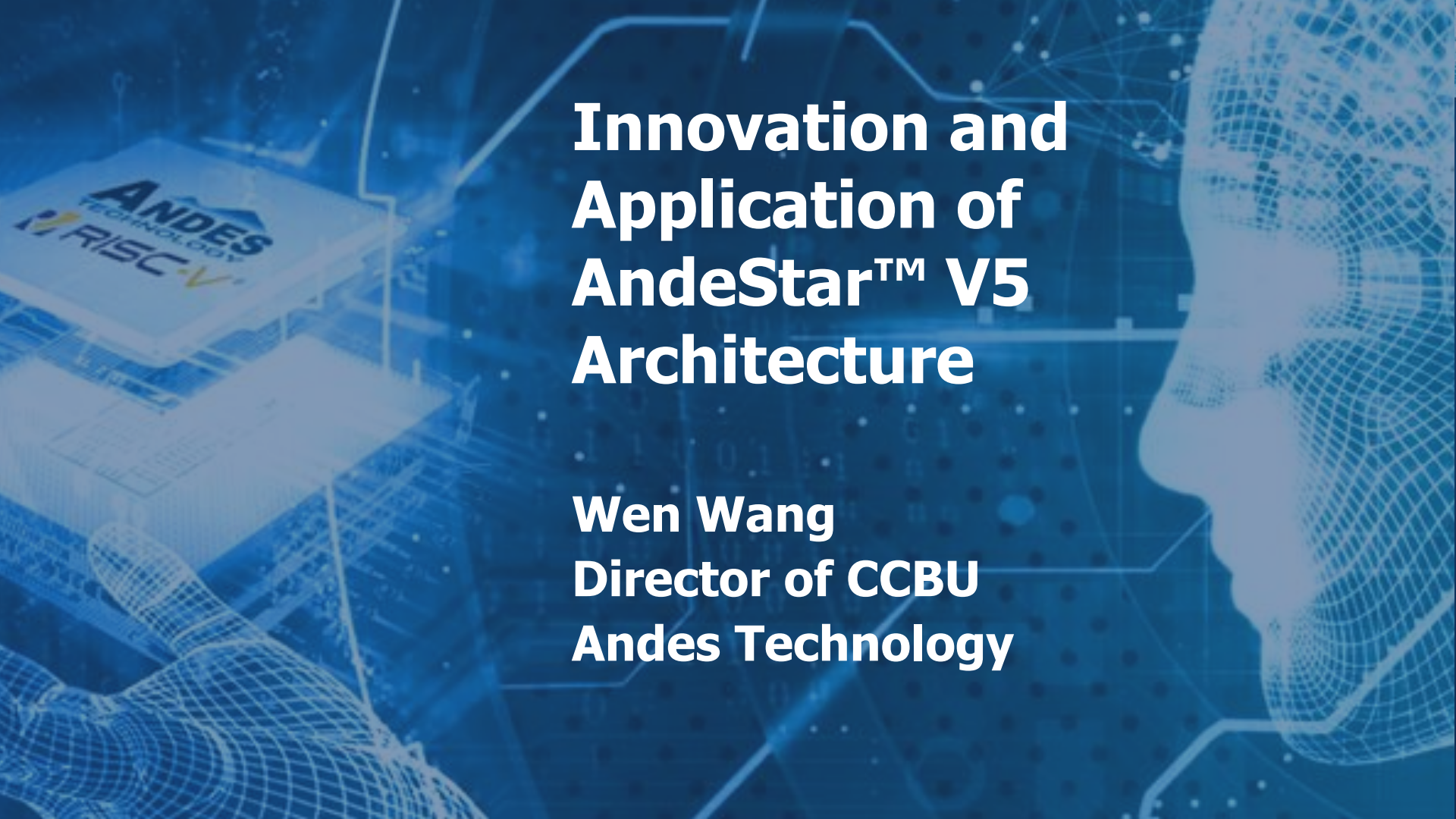


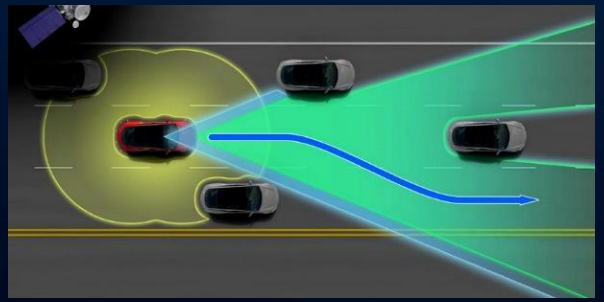
Innovation and Application of AndeStar™ V5 Architecture

Wen Wang
Director of CCBU
Andes Technology

前言

- AI is changing the world as smart phone did in the last decade
- IoT is connecting the world; everything is close at hand

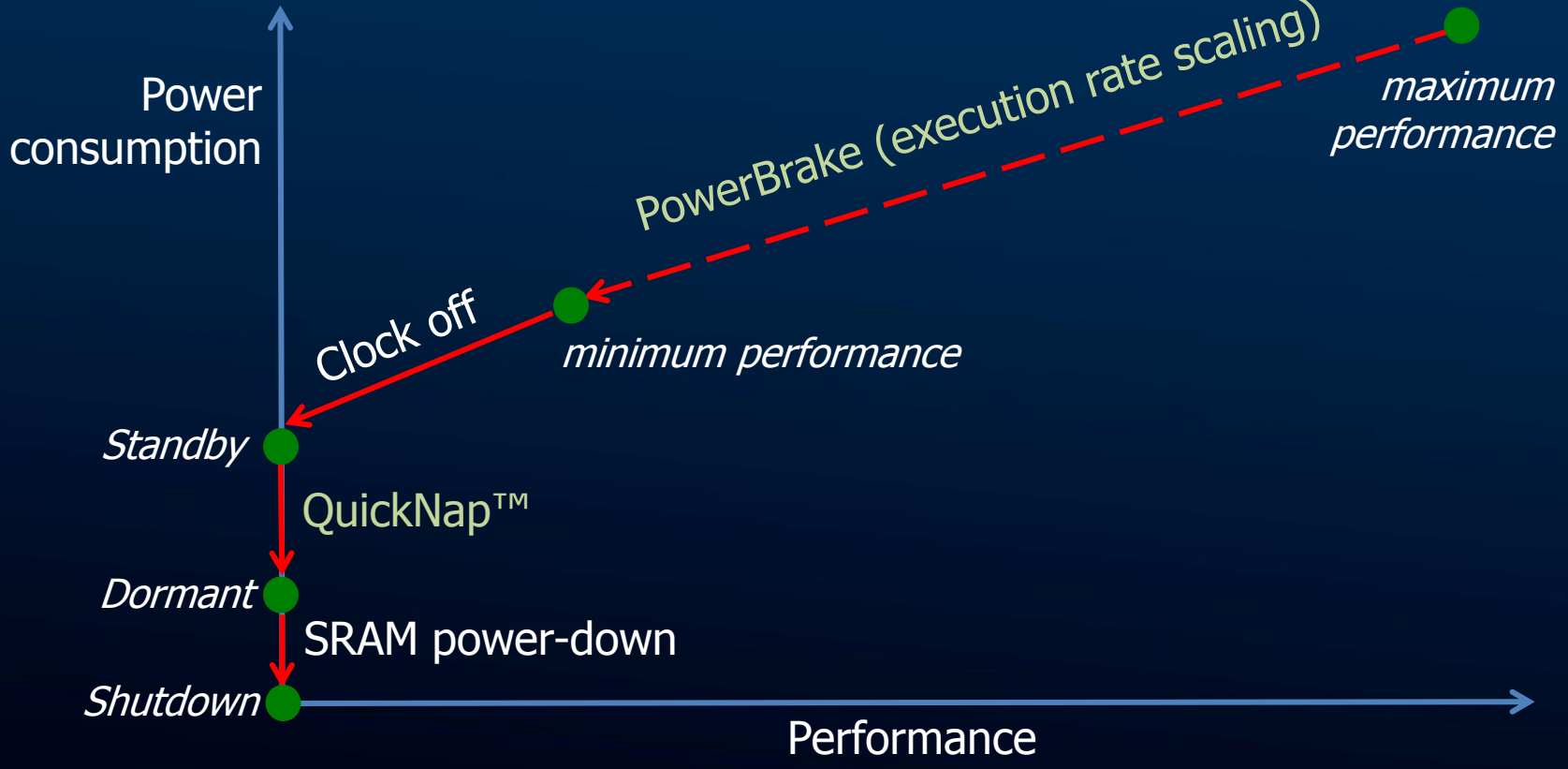
→ AIoT is the future; AIoT is the life



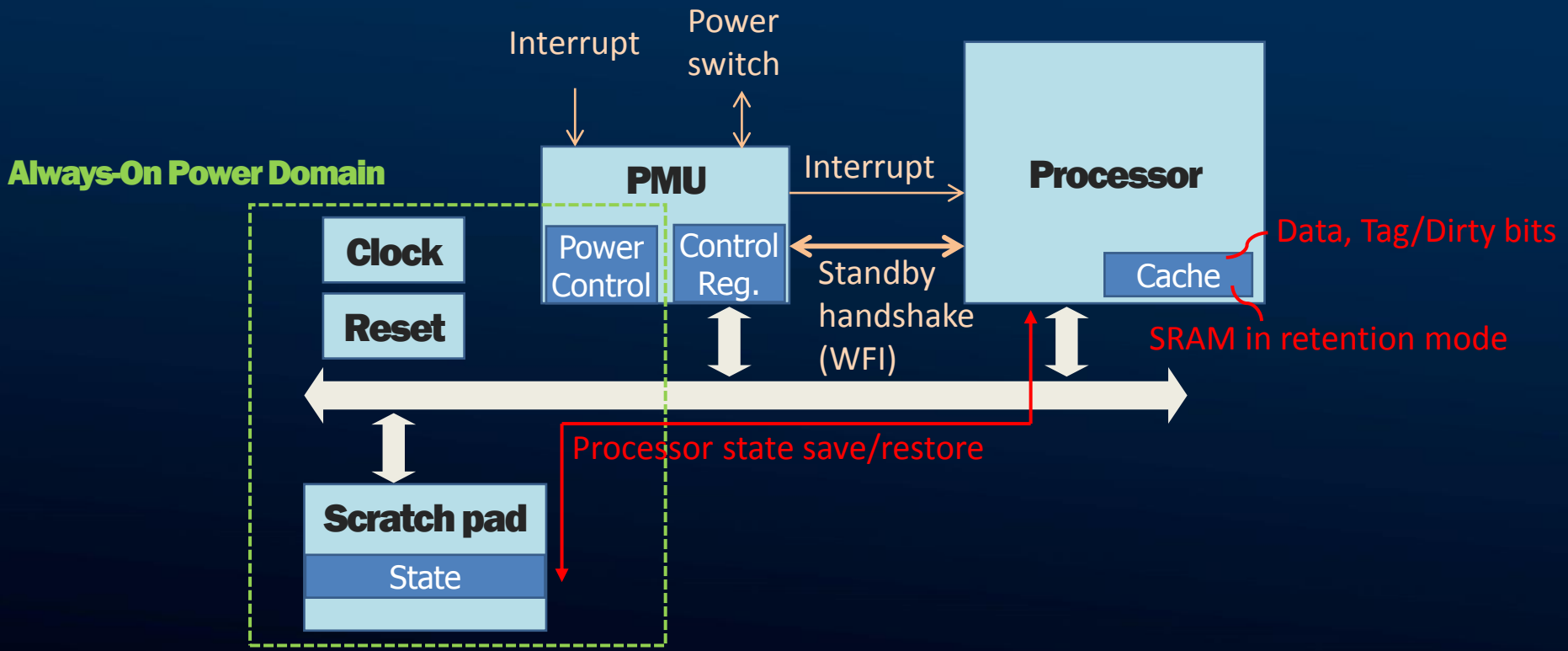
面臨的設計議題

- **High power efficiency and flexible power management → Power management**
  - For wearable and portable devices, long life time with battery
- **Efficient common media processing → RISC-V P-extension**
  - For voice, beam forming, slow image
- **Domain-specific acceleration → Andes Custom Extension**
  - For those requiring to process higher volume of data

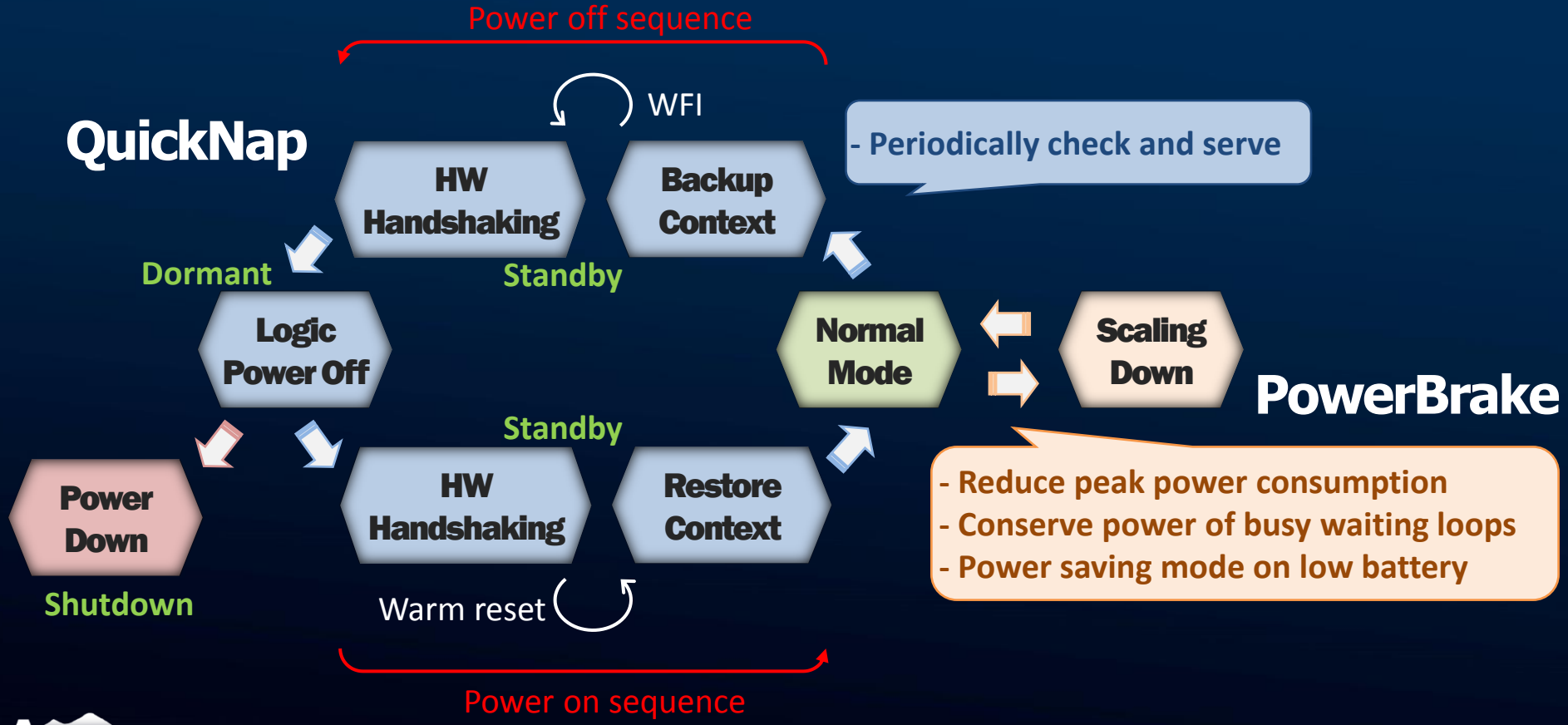
電源管理


Power consumption
PowerBrake (execution rate scaling)
maximum performance
Clock off
minimum performance
Standby
QuickNap™
Dormant
SRAM power-down
Shutdown
Performance

QuickNap™


Interrupt
Power switch
Always-On Power Domain
PMU
Interrupt
Processor
Clock
Reset
Power Control
Control Reg.
Standby handshake (WFI)
Cache
Data, Tag/Dirty bits
SRAM in retention mode
Processor state save/restore
Scratch pad
State

電源管理應用流程

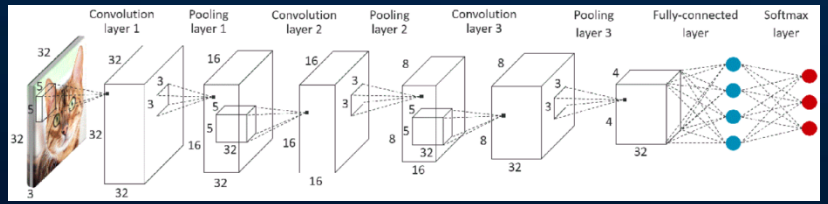

Power off sequence
WFI
QuickNap
HW Handshaking
Backup Context
- Periodically check and serve
Dormant
Standby
Logic Power Off
Normal Mode
Scaling Down
PowerBrake
Standby
Power Down
HW Handshaking
Restore Context
- Reduce peak power consumption
- Conserve power of busy waiting loops
- Power saving mode on low battery
Shutdown
Warm reset
Power on sequence

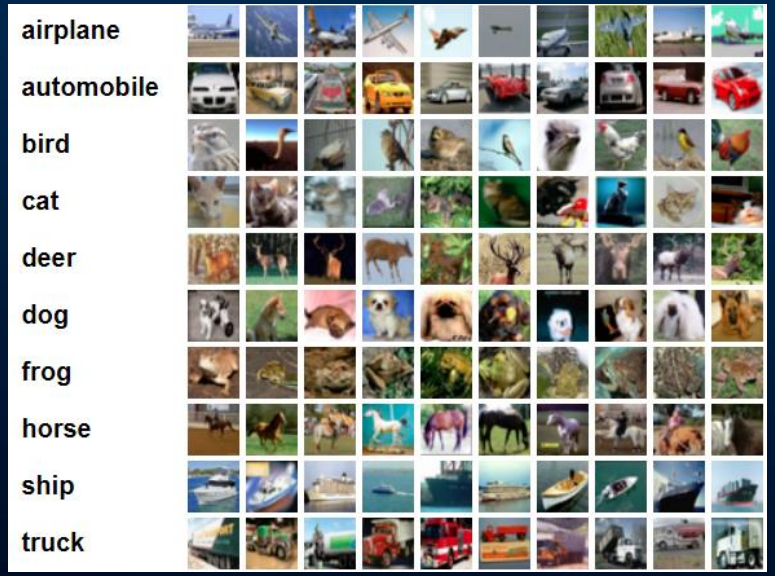
RISC-V P-擴展

- **Packed SIMD/DSP extension, currently a draft proposal**
- **Donated by Andes based on its popular V3 DSP ISA**
  - Enhanced with new instructions
  - Target efficient media processing based on GPR such as audio, voice and slow image
- **Details:**
  - Use RV32 and RV64 GPRs
  - Single instruction, multiple data: 8b, 16b, 32b element sizes
  - Fixed-point and integer data types
  - Saturation and rounding
  - Compiler and C language friendly (supported by intrinsic functions)
- **Optimized Andes libdsp for common DSP operations**
- **Optimized Andes libnn for neural network operations**

P-擴展:卷積神經網路應用實例1

■ CIFAR-10: image classification





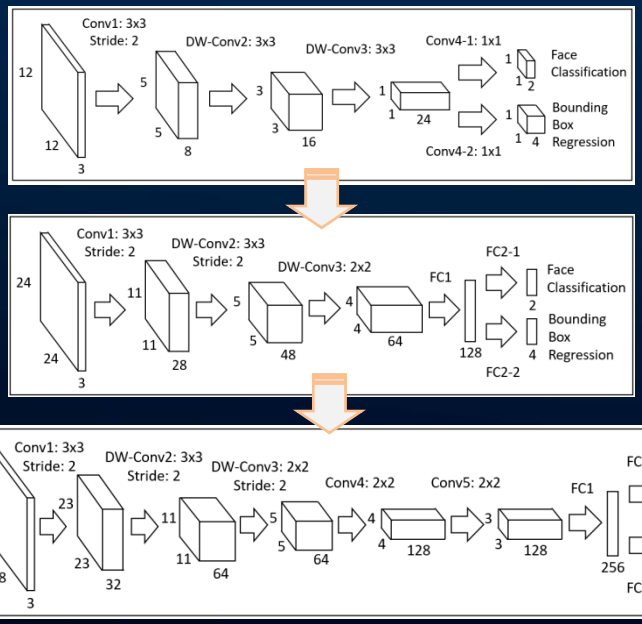
Overall speedup: 10.7x

Based on source C code, we invoke optimized functions in libnn, use some intrinsic functions to directly access to the DSP instructions with the help of DSP-capable compiler.

P-擴展:卷積神經網路應用實例2

CNN application: face detection

- Modified CNN (MTCNN), with model reduction and depthwise convolution



P-Net
Proposal network
obtain candidate windows

R-Net
Refine network
reject false candidate

O-Net
Output network

Overall speedup: 7.64x

應用實例常用的P-擴展指令

Most used instructions

kmada	SIMD Saturating Signed Multiply Two Halfs and Two Adds *32 = 32 + 16x16 + 16x16* *32 = 32 + 16x16 + 16x16 (2 sets with RV64)*
maddr32	Multiply and Add to 32-Bit Word *32 = 32 + 32x32 (low part)*
sunpkd8(x)(y)	Signed Unpacking Bytes x & y, xy = {10, 20, 30, 31, 32} *32 = {SE(16), SE(16)}*
pk(b/t)(b/t)32	Pack Two 32-bit Data from Bottom/Top Half (RV64 only) *64 = {32, 32}*
sclip32	SIMD 32-bit Signed Clip Value *6 = CLIP(32), plus overflow flag if saturation is performed*

客製化指令

- **Common misconception about "CPU"**
  - "CPU isn't suitable for complex computations or wide I/O"
- **RISC-V allows custom extensions for domain-specific acceleration (DSA)**
  - Andes Custom Extension (ACE)
  - All ACE instructions are assigned within the CUSTOM-3 space of the RISC-V opcode

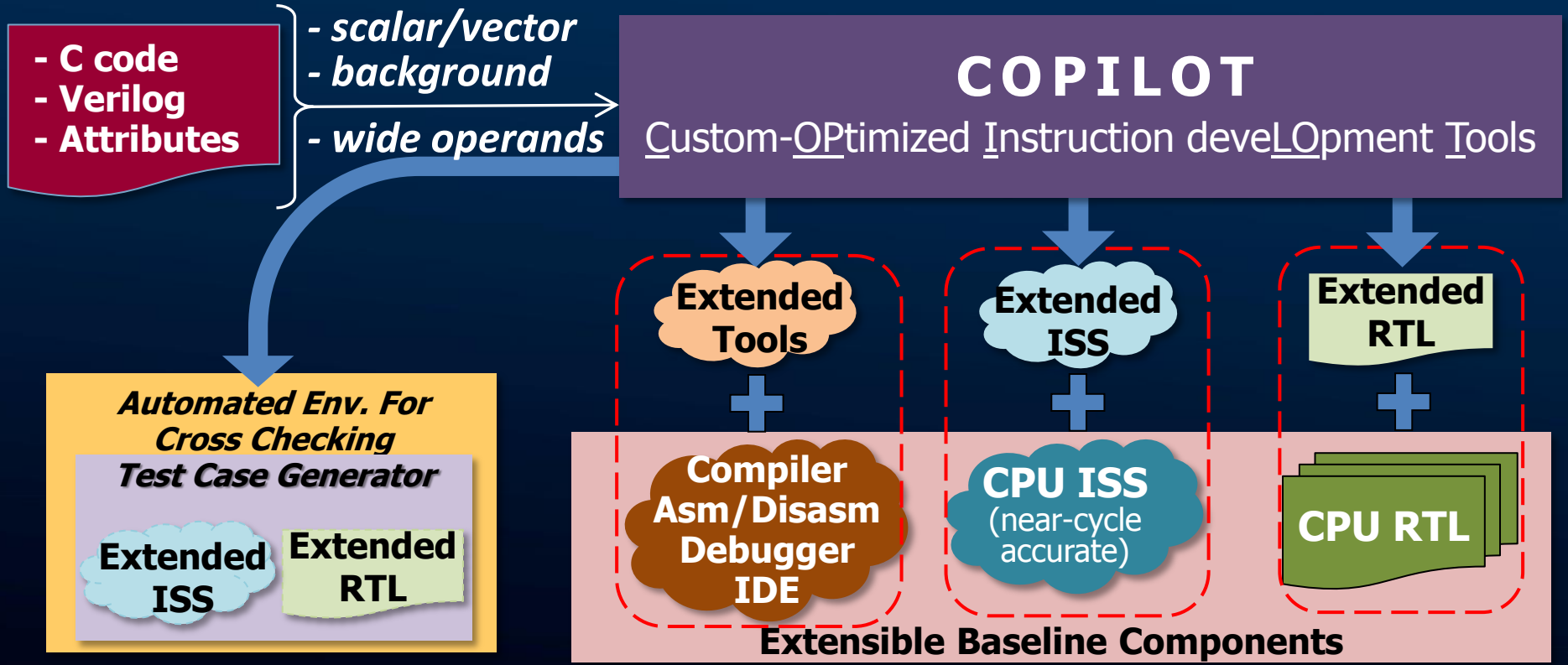
Andes Custom Extension (ACE)

- A simple description script to describe custom operations
- A tool (COPILOT) to generate most housekeeping works for users:

  - Opcode assignment: automatic by default
  - All required tools (compiler, assembler, disassembler and debugger), and simulator (C or SystemC)
  - RTL code for instruction decoding, operand mapping and accesses, dependence checking, result gathering, etc.
  - Verification environment/patterns and error reporting

Fast turnaround time!

ACE框架


- C code
- Verilog
- Attributes
- scalar/vector
- background
- wide operands
COPILOT
Custom-OPtimized Instruction develOpment Tools
Automated Env. For Cross Checking
Test Case Generator
Extended ISS
Extended RTL
Extended Tools
Extended ISS
Extended RTL
Compiler Asm/Disasm Debugger IDE
CPU ISS (near-cycle accurate)
CPU RTL
Extensible Baseline Components

ACE特色摘要

Items	Description	
Instructions	scalar	single-cycle, or multi-cycle
	vector	for loop, or do-while loop
	background option	retire immediately, and continue execution in the background. Applicable to scalar and vector.
Operands	standard	immediate, GPR, baseline memory (thru CPU)
	custom	- ACR (ACE Register), ACM (ACE Memory), ACP (ACE Port) - With arbitrary width and number - Operands can be "implied" to save opcode

如何設計ACE指令

- ACE description: describe instruction interfaces (operands) and behaviors
- ACE RTL design: implement RTL code for instruction operations in concise Verilog

ACE設計範例: madd32

ACE Description Script:

Instruction name

Operand list

```
insn madd32 {
  op = {io gpr acc, in gpr dat, in gpr coef};
  csim = %{
    acc+= (dat & 0xffff) * (coef & 0xffff)
        + (dat>>16)      * (coef>>16);
  %};
  latency = 1;
};
```

madd32.ace

Automatically generate intrinsic function:
uint32 ace_madd32(uint32, uint32, uint32)

Instruction semantics in C for Instruction Set Simulator (ISS)

Instruction latency in CPU cycles

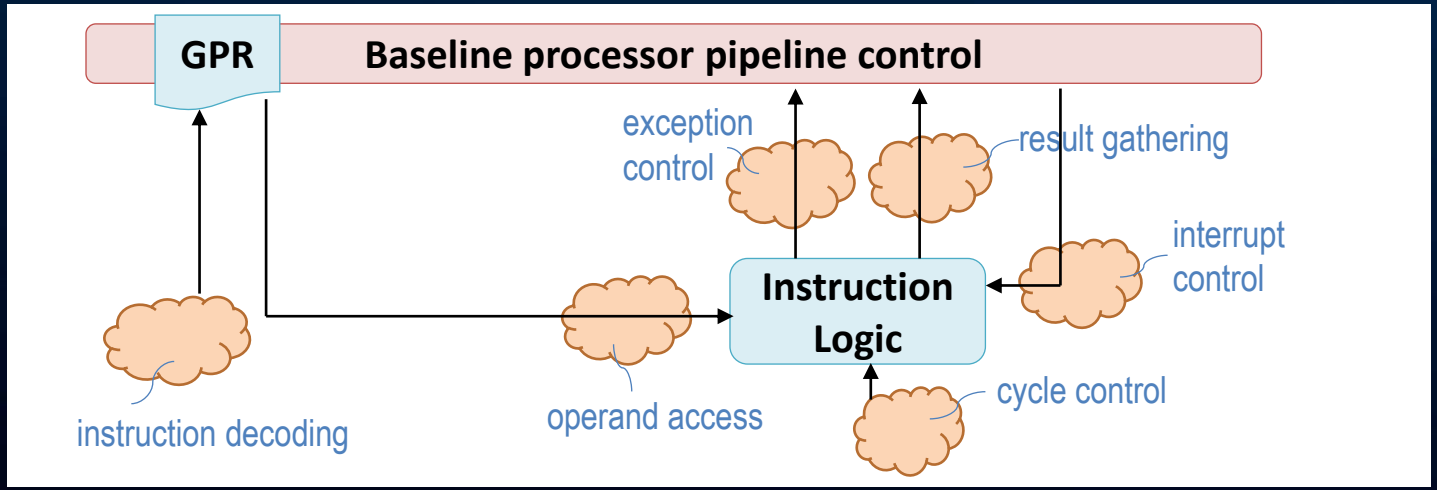
ACE設計範例: madd32

ACE RTL Design Language: (a Verilog-based but concise form)

madd32.v

```
// ACE_BEGIN: madd32
assign acc_out = acc_in
               + dat[15:0]  * coef[15:0]
               + dat[31:16] * coef[31:16];
// ACE_END
```

instruction-specific logic
(no declarations needed for operands and module I/O)



ACE自定寄存器設計範例

ACE Custom Register (ACR)

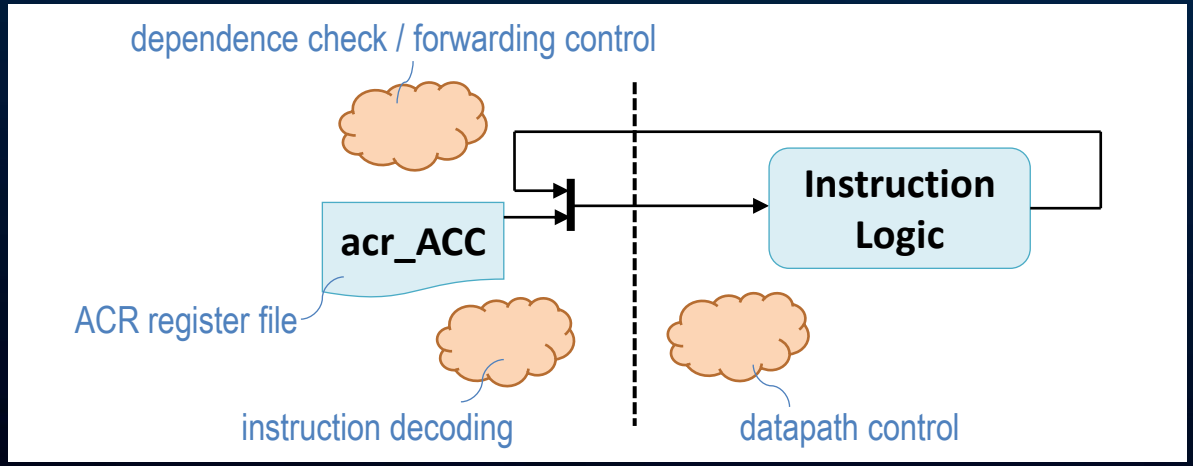
Register name

Example:

```
reg ACC {
  number= 4;
  width= 64;
};
```

→ Total 4 entries

→ 64-bit data width



ACE自定記憶體設計範例

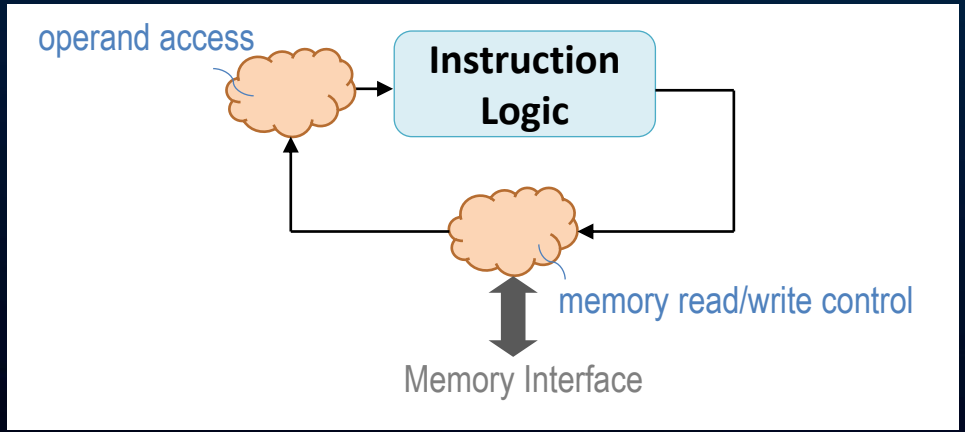
Andes Custom Memory (ACM)

Memory name

Example:

```
ram XMEM {
  interface= SRAM;
  width= 32;
  address_bits= 12;
};
```

→ SRAM interface
→ 32-bit data width
→ 12-bit address space



vmadd32: Vectorizing madd32

Vector Instruction

```
vec insn vmadd32 {
  operand= {io gpr acc,
            in XMEM dat, in YMEM coef,
            imm5 cnt};
  loop_type= repeat(cnt);
  stride<dat>= 1;
  csim= %{
      ...
  %};
  latency= 1;
};
```

```
ram XMEN { //same for YMEM
  interface= SRAM;
  width= 32;
  address_bits= 12;
};
```

→ for loop, repeat "cnt" times

→ Memory address automatically increases 1 for next iteration

→ per-iteration operation and latency

csim & RTL: same as scalar version !!!

```
// ACE_BEGIN: vmadd32
...
// ACE_END
```

→ per-iteration logic

bvmadd32: Backtorizing madd32

Background Vector Instruction

```
vec bg_insn bvmadd32 {
  operand= {io gpr acc,
            in XMEM dat, in YMEM coef,
            imm5 cnt};
  loop_type= repeat(cnt);
  stride<dat>= 1;
  csim= %{
      ...
  %};
  latency= 1;
};
```

```
ram XMEM { //same for YMEM
  interface= SRAM;
  width= 32;
  address_bits= 12;
};
```

Everything else:
same as foreground version !!!

```
// ACE_BEGIN: bvmadd32
...
// ACE_END
```

Background instruction is executed in parallel with all other instructions

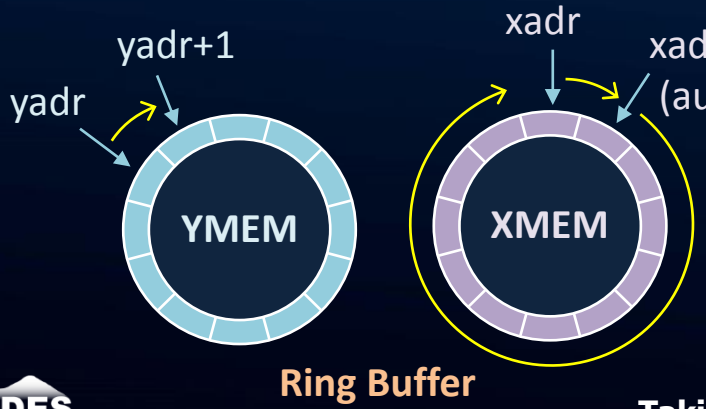
進階範例: 環形緩衝區使用

post-update (ACE generated logic will get it done!)

```
insn madd32rb {
  op= {io gpr acc,
       in XMEM@xadr:u dat, in YMEM@yadr:u coef};
  csim= %{
     acc+=(data & 0xffff)* (coef & 0xfff)
         +(data >> 16)    * (coef >> 16);
  %};
};
```

address from custom register

```
ram XMEM {    //same for YMEM
  interface= SRAM;
  width= 32;
  address_bits= 12;
};
reg xadr {  //same for yadr
  number= 4;
  width= 12;
};
```



xadr+1 for next use
(auto-increment and auto-update)

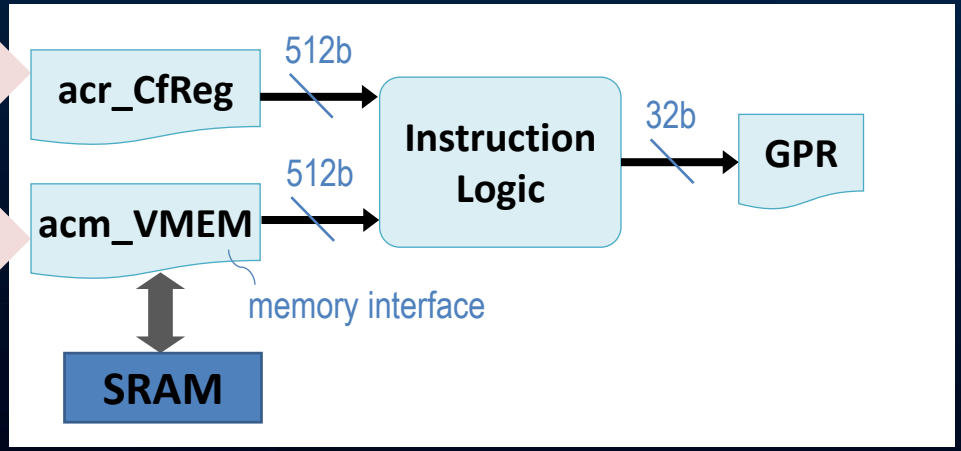
the address is wrapped-around to 0 on overflow

Ring Buffer

進階範例: 64個8-bit數據向量的內積

```
insn innerp {
  op= {out gpr IP, in CfReg C, in VMEM V};
  csim= %{
    IP= 0;
    for(uint i= 0; i<64; ++i)
        IP+= ((C >>(i*8)) & 0xff) *
             ((V >>(i*8)) & 0xff);
  %};
  latency= 3;        //enable multi-cycle ctrl
};
reg CfReg {          //Coef Registers
    num= 4;
    width= 512;
};
ram VMEM {           //data memory
    interface= sram;
    address_bits= 3;   //8 elements
    width= 512;
};
```

```
//ACE_BEGIN: innerp
assign IP= C[ 7:0]    * V[ 7:0]
         + C[15:8]    * V[15:8]
           . . .
         + C[511:504] * V[511:504];
//ACE_END
```



ACE的優勢

- **Users focus instruction semantics, not CPU pipeline**
- **Housekeeping tasks are offloaded to COPILOT**
  - opcode selection and instruction decoding
  - operand mapping/accesses/updates
  - dependence checking
- **Comprehensive support**
  - Powerful instruction semantics: vector, background, wide operands
  - Auto-generation of verification environment, development tools and RTL code
- **ACE unlocks RISC-V's potential for DSA**

Andes 提供客製化服務

- **Andes Custom Extension**
  - COPILOT tool license
- **Andes Custom Computing BU (CCBU)**
  - Analysis of customization algorithm
  - Evaluation of customization instruction/IP
  - Deliverable of customized IP
  - Maintenance for customized IP
- **New business model for customizing RISC-V CPU for you**

結語

- **More and more AIoT applications are emerging with next generation 5G networking**
  - Smart home, office
  - Intelligent retail, manufacturing, medical treatment
- **Power consumption and computation capabilities are among the most critical issues**
- **Andes provides innovated solutions helped customers to innovate their SoC**

Thank you