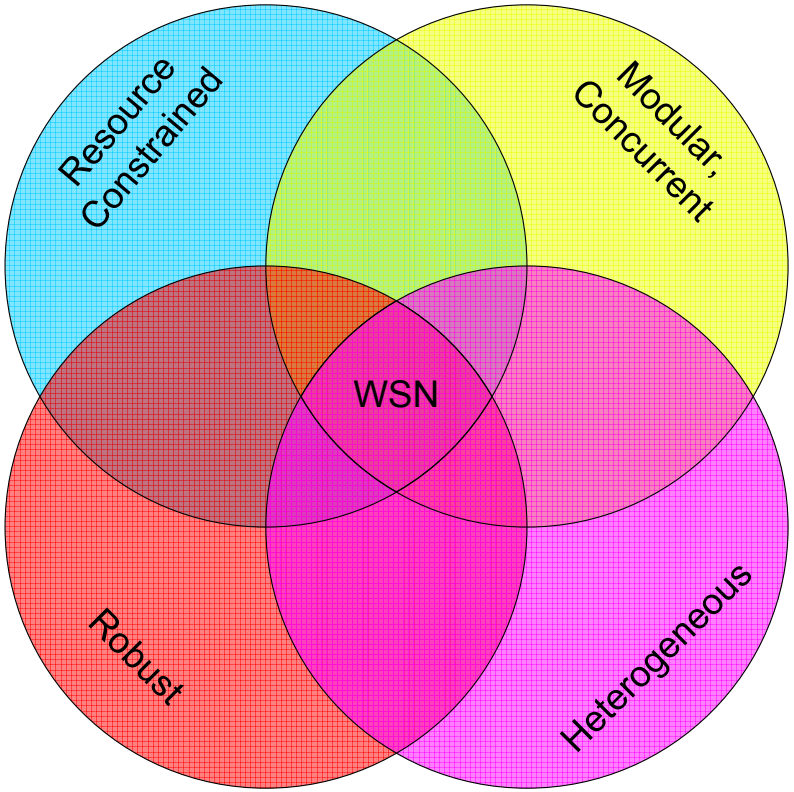
Darjeeling, a Feature-Rich VM for the Resource Poor

Niels Brouwers, Peter Corke, and Koen Langendoen

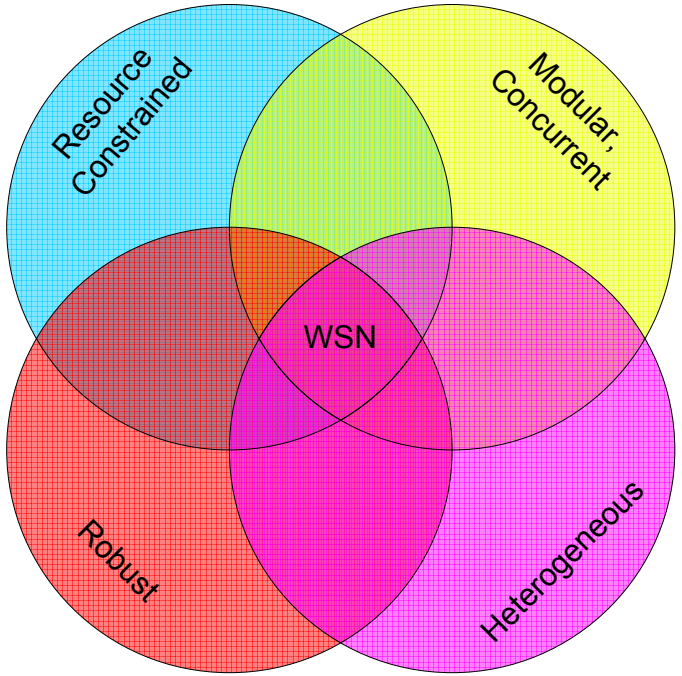
Introduction



TU Delft

Introduction

- Operating systems
  - Contiki, TinyOS
  - Native code (C/NesC)
  - Efficient concurrency
- Virtual Machine
  - Interpreted (Java)
  - Robust, feature-rich
  - Time/space overhead



TU Delft

Introduction

- We would like a VM that
  - Enables a high-level language (Java)
  - With an acceptable overhead
- Memory efficiency highest priority
  - RAM
  - Program flash

Previous work

	Feature rich	Resource efficient	Open source
Sentilla	yes	yes	no
TinyVM (LeJOS)	no	yes	yes
NanoVM	no	yes	yes
Taka Tuka	yes	yes	no
VM*	yes	yes	no
MoteRunner	yes	yes	no
Java Card	no	yes	no
Squawk (Sun Spot)	yes	no	yes

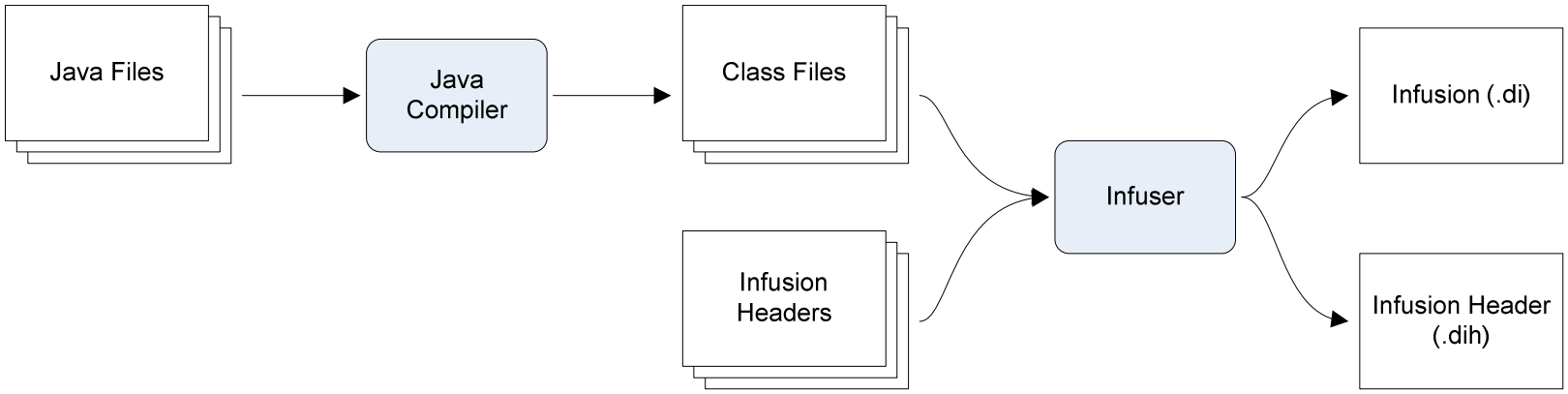
TU Delft

Darjeeling

- Feature rich
  - Threading, synchronisation
  - Compacting garbage collection
- Resource efficient
  - Can run on devices with 2-10k RAM, ~64k flash
  - But, no support for 64-bit or floating point, reflection
- Portable
  - Contiki, TinyOS, FOS
- Open source

Infuser tool

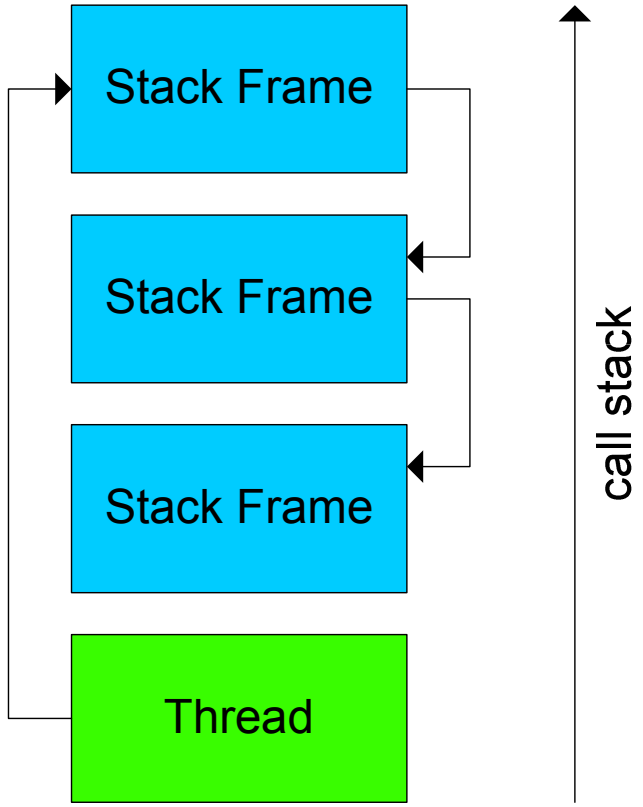
- Performs static linking
- Byte code post-processing
- Reduces code footprint by removing entity names



TU Delft

Linked stack architecture

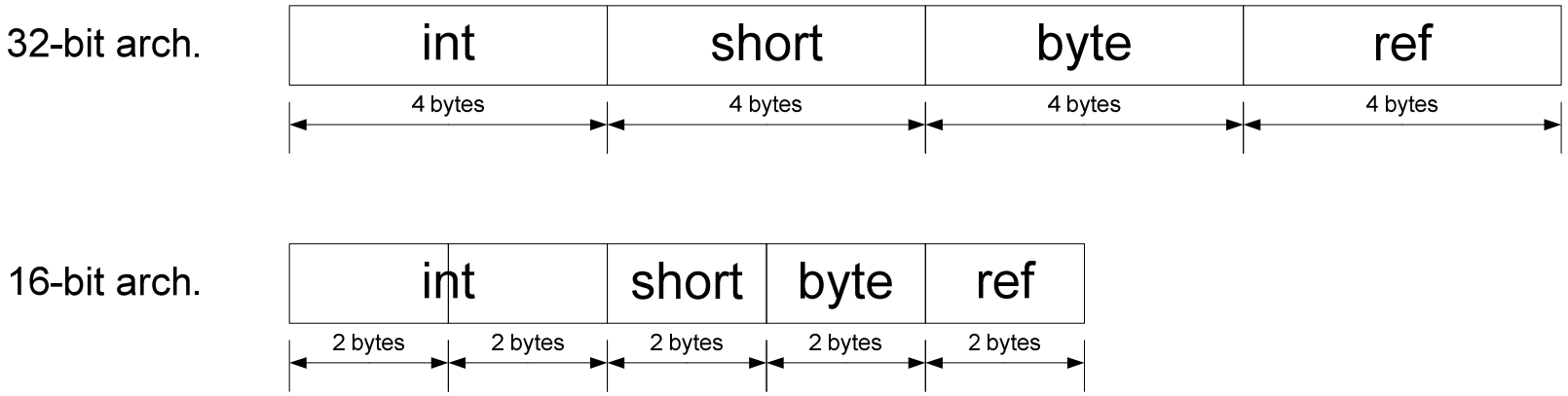
- Light-weight threads
  - Stack frames are heap objects
  - On demand stack space allocation
  - Min memory usage: ~45 bytes



TU Delft

16-bit architecture

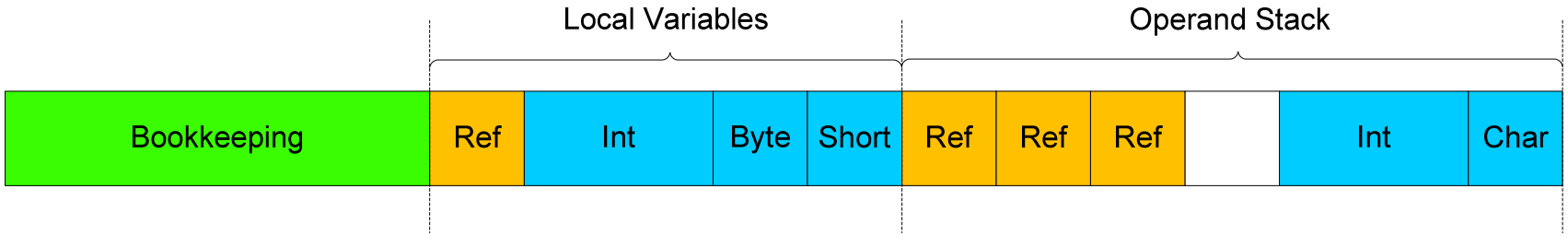
- Custom instructions for 16-bit data types
- Saves stack space (operand stack, local variables)
- Arithmetic optimisation



TU Delft

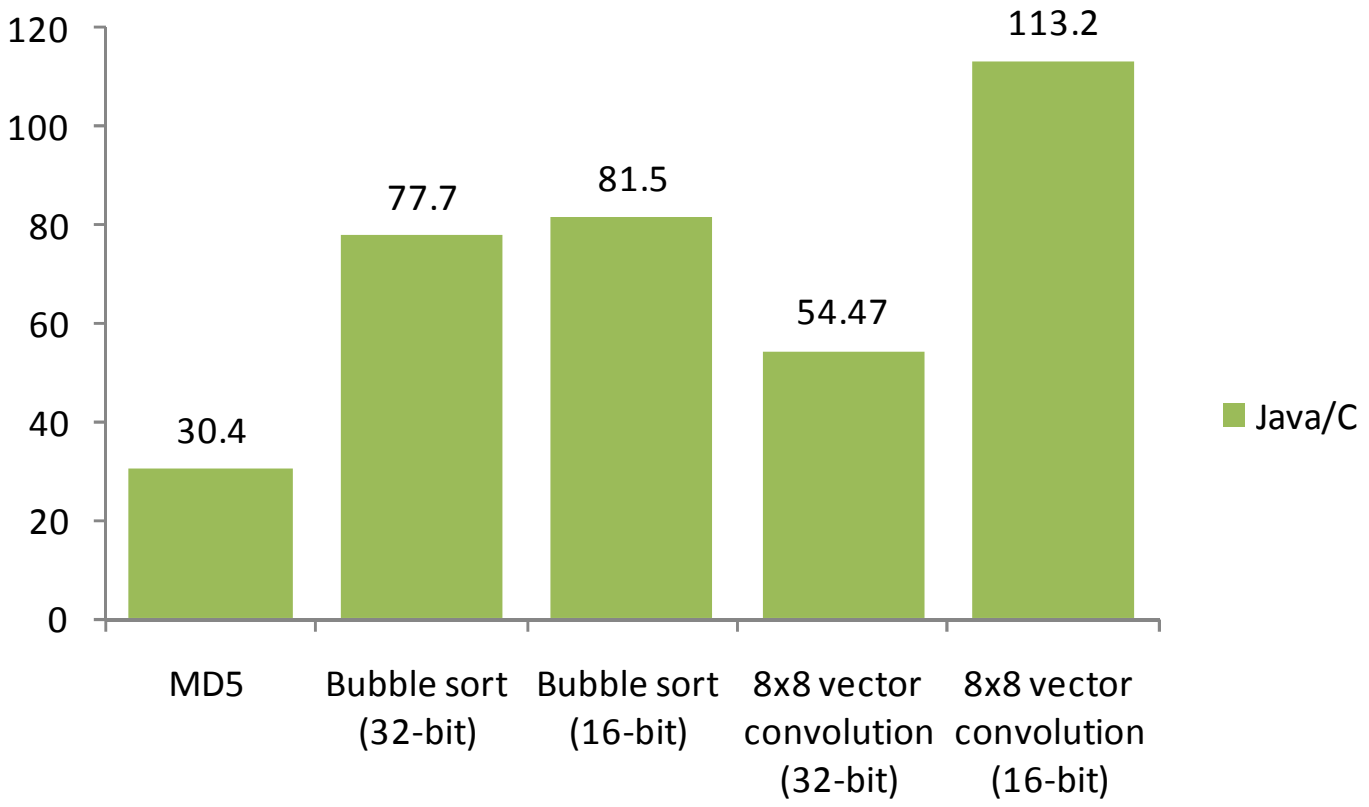
Double-ended stack

- Precise, compacting garbage collector
  - Compaction requires identifying references on the operand stack
  - Darjeeling uses two logical stacks



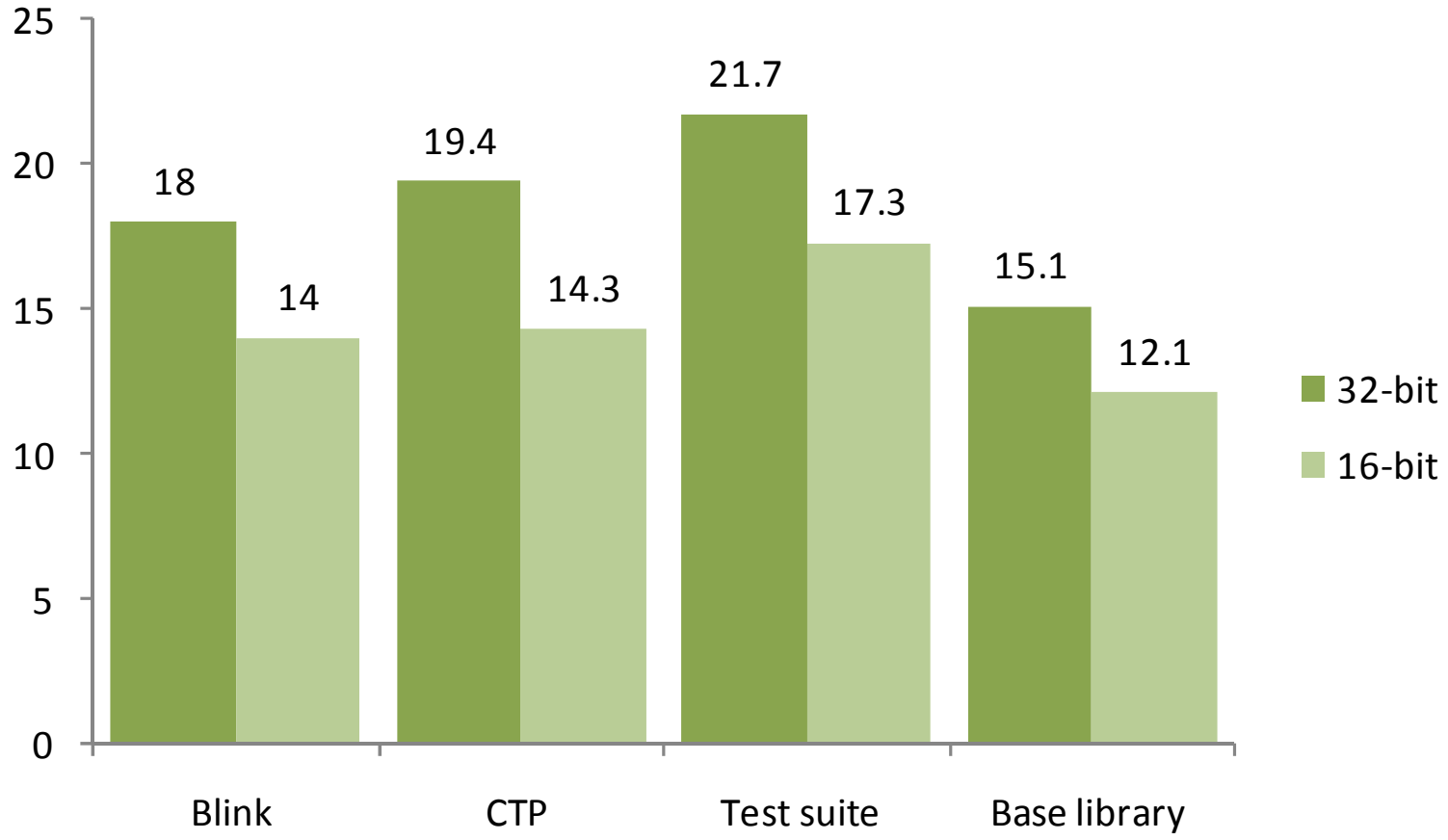
TU Delft

Performance: Java vs C



TU Delft

Memory consumption: Stack Space



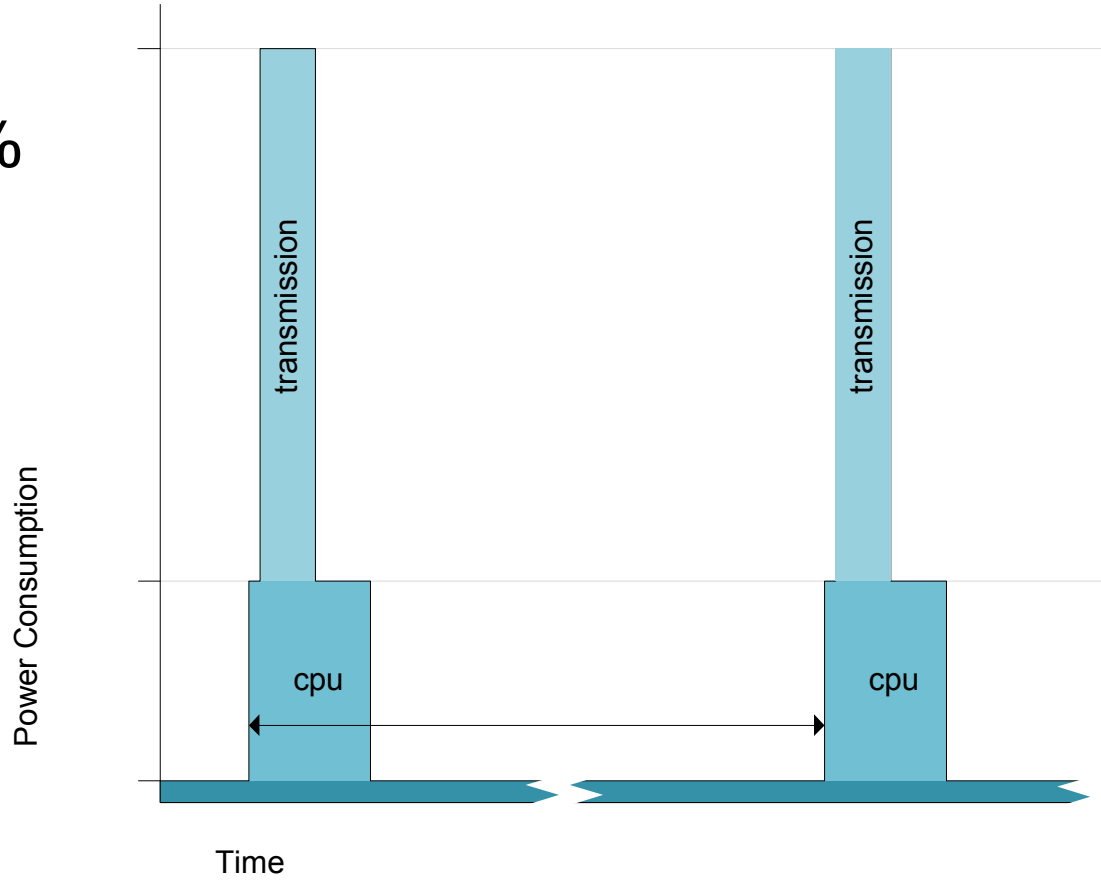
TU Delft

Springbrook

- Rewrite of a real-world application
- CTP implementation in Java
  - 22 classes, ~1200 sloc
  - <8kB in program flash
  - <2kB RAM
  - 5 threads

Springbrook power consumption

- CPU on-time increased by 50%



TU Delft

Supported platforms

	Tnode	Tmote Sky	Fleck3/Fleck3B
MCU	Atmega128	MSP430	Atmega128(1)
Architecture	8-bit	16-bit	8-bit
RAM	4kB	10kB	4/8kB
Radio	CC1000	CC2420	nRF905
OS	TinyOS	Contiki	FOS
Concurrency	events	protothreads	threads

Conclusions

- Darjeeling is an embedded VM
  - Feature rich
  - Memory efficient
  - Open source
- Suitable for real-world applications
- Performance overhead acceptable for many applications

Future work

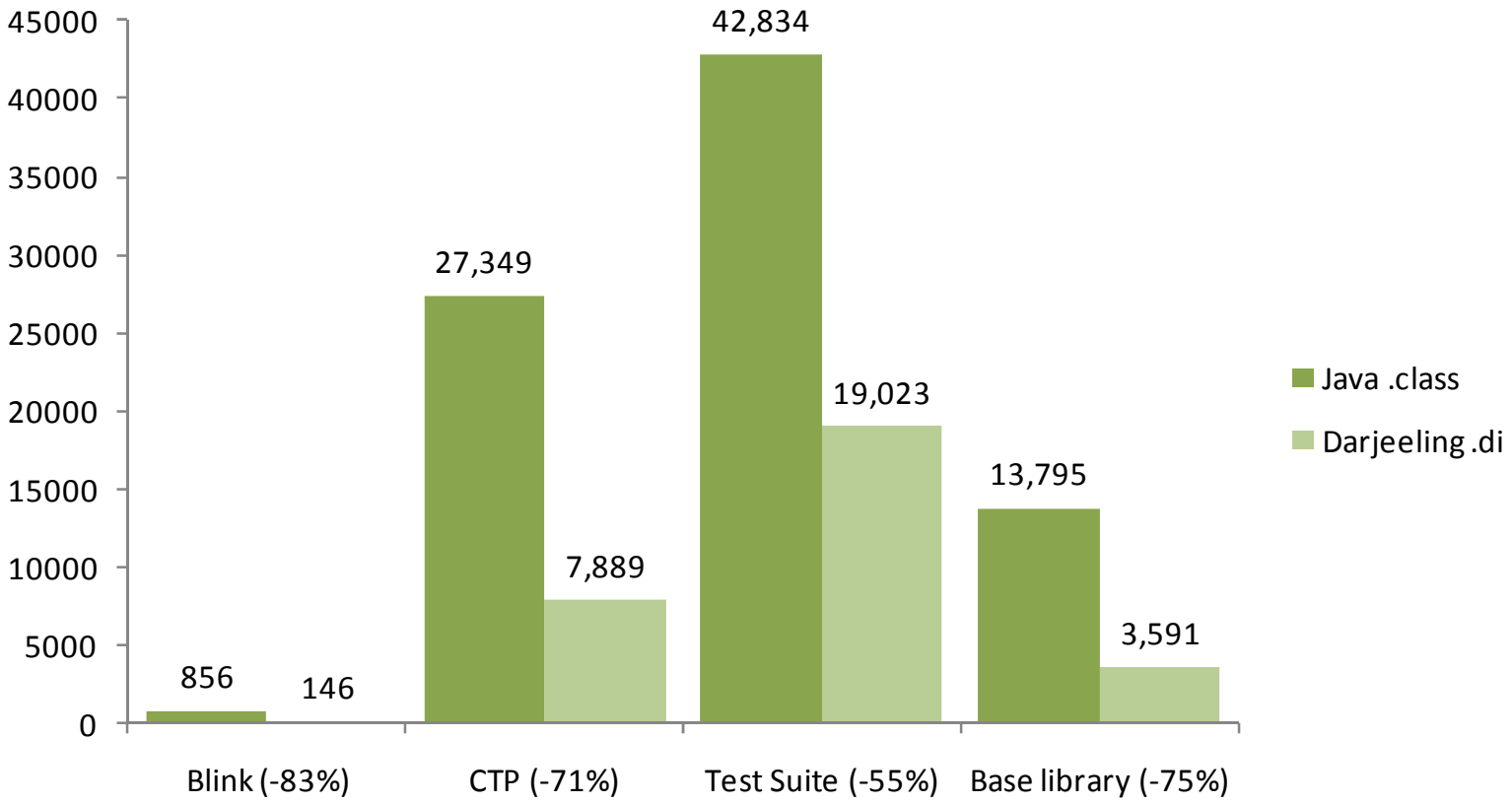
- CLDC 1.0 compliance
  - Support for the `long` data type
  - Reflection
- Over-the-air loading
- Generational garbage collector ?
- Operating system independent ?

http://darjeeling.sf.net

Questions?

TU Delft

Code size reduction



TU Delft

Micro benchmarks: performance

Test	C	Java	Java/C	Instr./s
MD5	13.1s	399.7s	30.4	52,294
Bubble sort (32-bit)	0.3s	23.3s	77.7	60,618
Bubble sort (16-bit)	0.2s	19.3	81.5	71,526
8x8 vector convolution (32-bit)	9.1s	496.7s	54.47	65,454
8x8 vector convolution (16-bit)	4.6s	520.9s	113.2	71,512

Micro benchmarks: code size

	jar	di	reduction
Blink	856	146	83%
CTP	27,349	7,889	71%
Test Suite	42,834	19,023	55%
Base library	13,795	3,591	75%

Micro benchmarks: stack size (1)

	JVM	DVM	reduction
Blink	18	14	22%
CTP	19.4	14.3	27%
Test suite	21.7	17.3	20%
Base library	15.1	12.1	20%

Micro benchmarks: stack size (2)

	JVM	DVM	reduction
Blink	10	6	40%
CTP	11.4	6.3	45%
Test suite	13.7	9.3	32%
Base library	7.1	4.1	42%

Portability

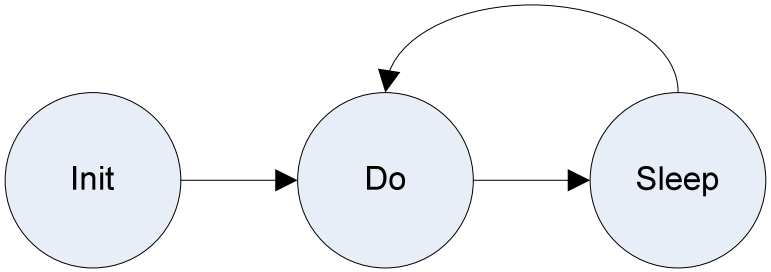
	Tnode	Tmote Sky	Fleck3/Fleck3B
MCU	Atmega128	MSP430	Atmega128(1)
Architecture	8-bit	16-bit	8-bit
RAM	4kB	10kB	4/8kB
Radio	CC1000	CC2420	nRF905
OS	TinyOS	Contiki	FOS
Concurrency	events	protothreads	threads

Portability: code complexity

	Tnode	Tmote Sky	Fleck3/Fleck3B
Main	225	359	86
Radio	69	92	69
Sensors	184	157	110
Total	478	608	265

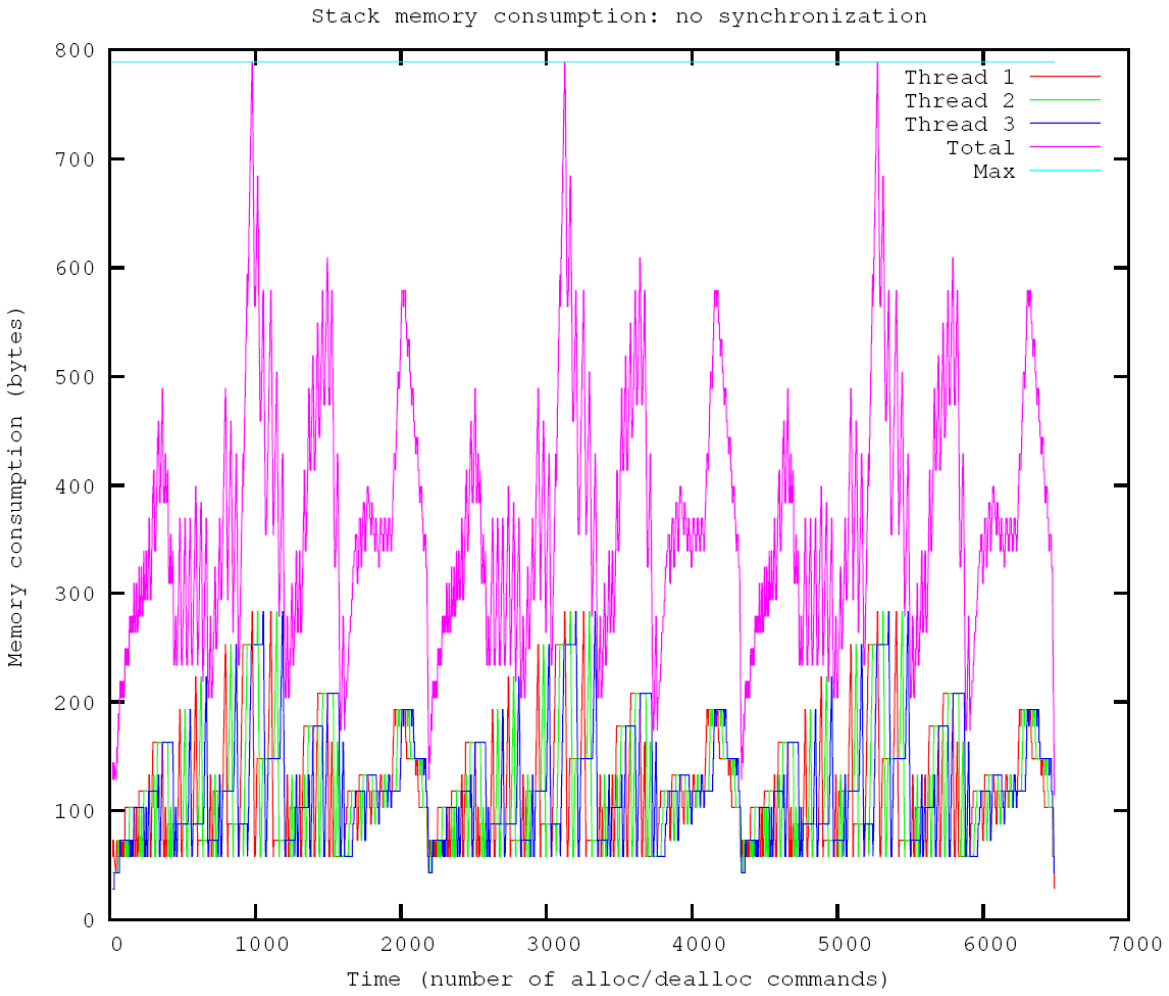
Linked stack architecture

- Test runs three threads, started at the same time
- Each thread performs a binary tree test three times, with a 10 millisecond sleep in between
- Let the heap manager log the *total stack size* of each thread as the program executes
- Simulates periodic work, common in WSN



TU Delft

Linked stack architecture

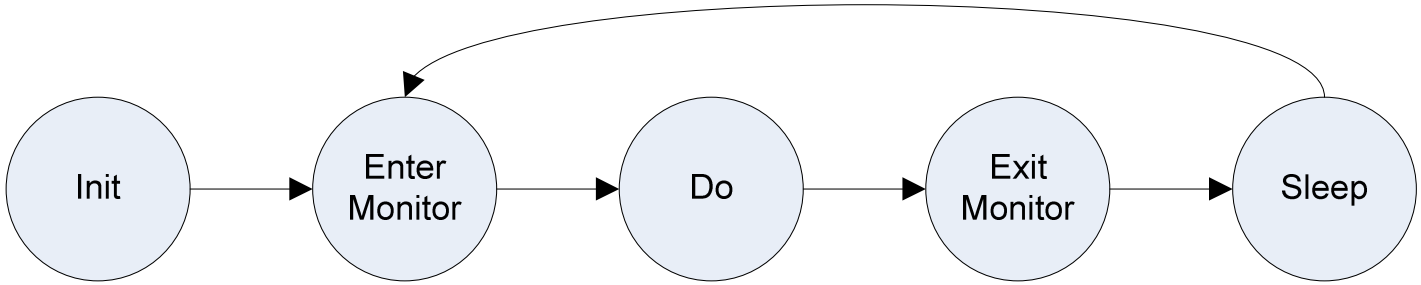


Peak memory consumption at 1335 bytes.

TU Delft

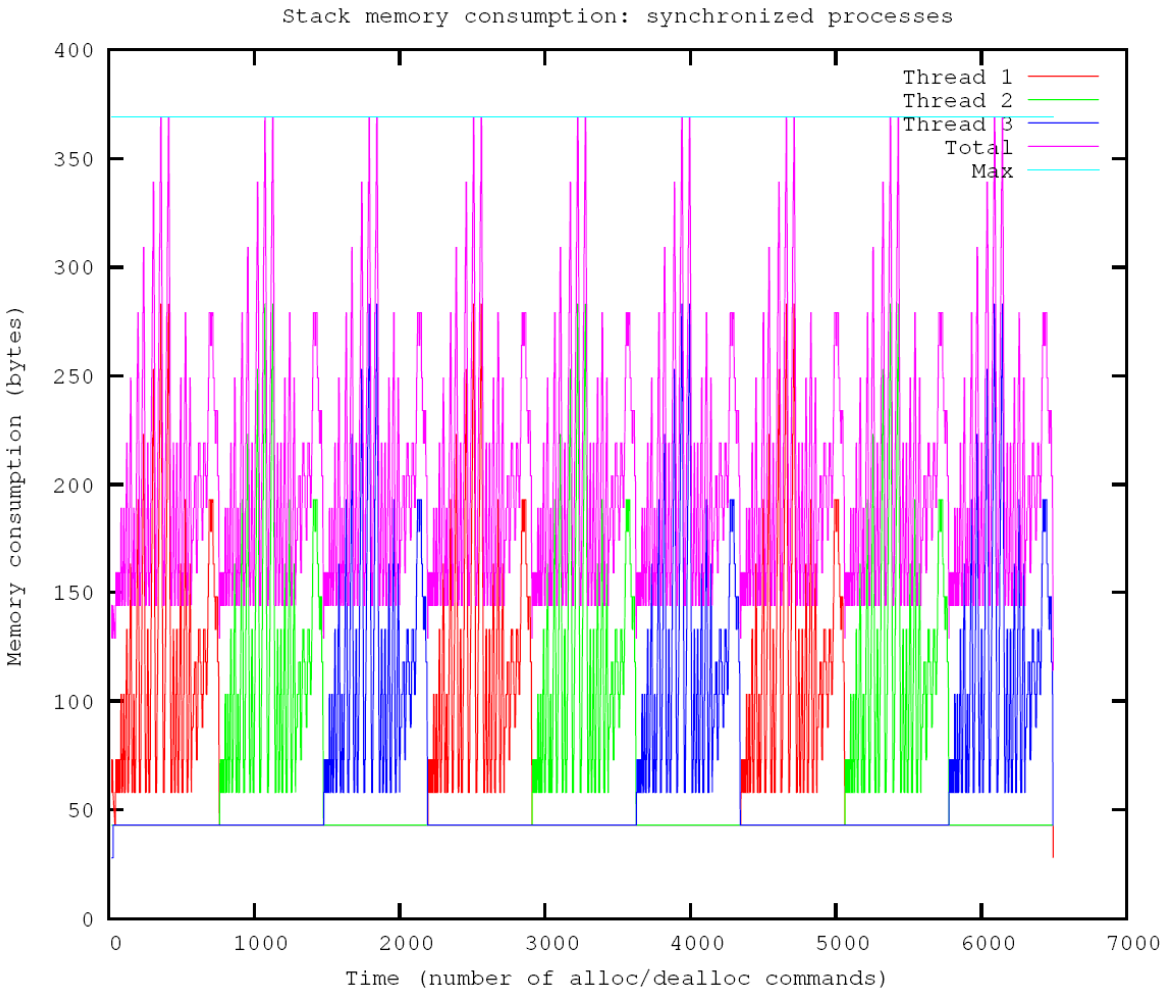
Linked stack architecture

- Synchronize threads
- Only one thread may perform work at a time
- Work is serialised automatically



TU Delft

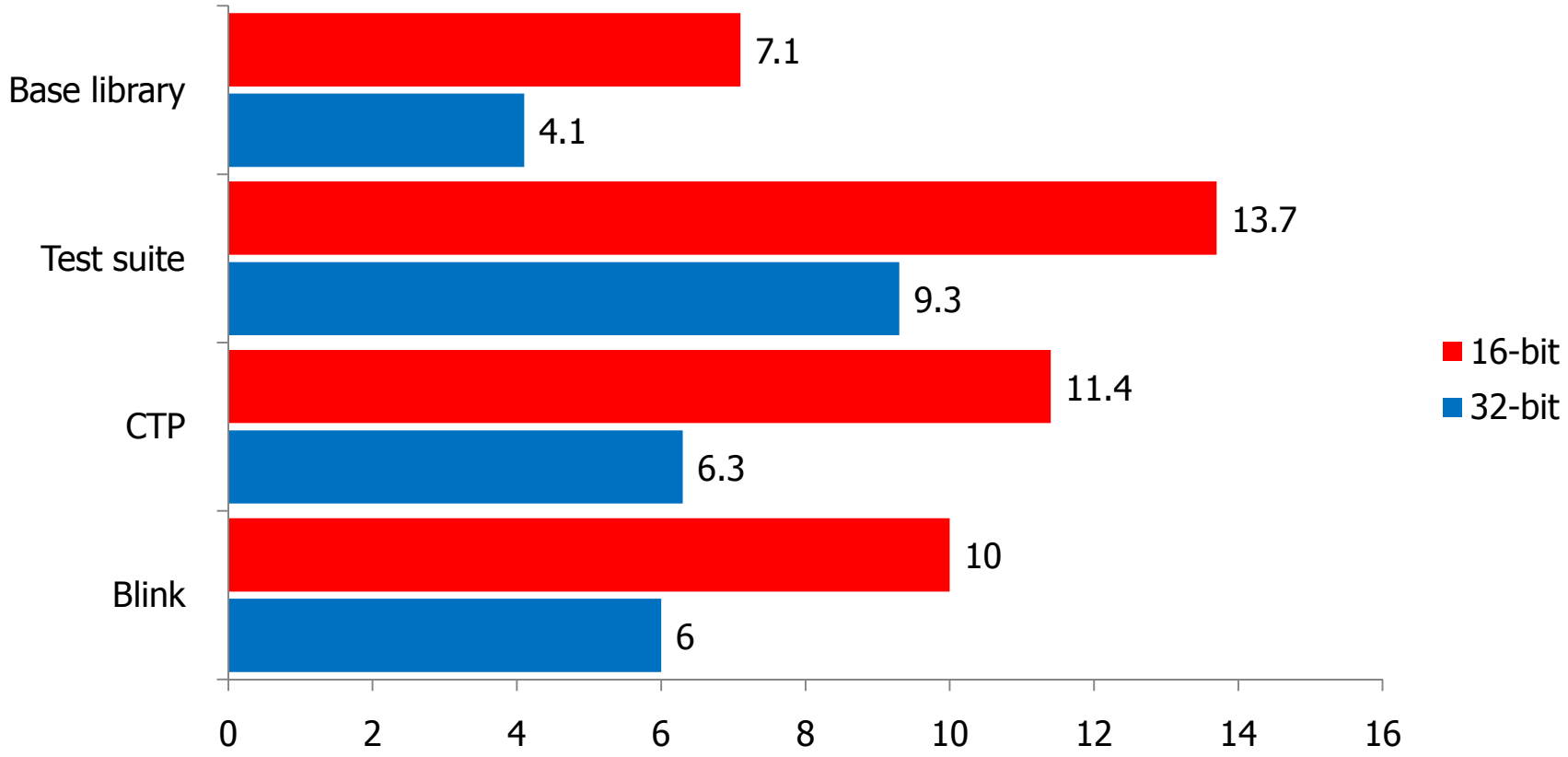
Linked stack architecture



Peak memory consumption down to 535 bytes.

TU Delft

Memory consumption: Stack Space



TU Delft