# A Decoupled Local Memory Allocator

Boubacar Diouf[1,2]

Can Hantaş[3], Albert Cohen[1,2], Özcan Öztürk[4], Jens Palsberg[5]

[1]INRIA

[2]École Normale Supérieure de Paris

[3]Georgia Institute of Technology

[4]Bilkent University

[5]UCLA

# Cache/Local Memories

GP CPUs and embedded systems usually have a small quantity of SRAM used to speed up the executed programs. The SRAM is generally configured as a:

## Cache

- Automatic cache management in hardware
- Efficient on general purpose CPUs

## Local Memory

- Supported by the user or the compiler
- Fast, predictible, power efficient, smaller area cost
- Many embedded processors, DSPs, GPUs, Cell SPU have local memories (LM)

How to efficiently allocate data to the local memory?

# The Link between LM and Register Allocations

## Decoupled register allocation

- Allocation (rely on maxlive, choose register residents) **NP-complete**
- Assignment (which register for which variable) **polynomial under SSA**

Decoupling: isolate the hard problem of allocation (spilling)

## Decoupled local memory allocation

- Allocation (rely on maxsize, choose local-memory residents) **NP-complete**
- Assignment (for each array block, where it should reside in the LM)
  - Sufficient condition (criterion)?
  - Complexity?

This link, discovered by Fabri [Fab'79], thirty years ago, has been under-exploited

# Outline

Introduction

Weighted Interval Graph Coloring

LM Allocation through NSP WIG Coloring

Experimental Evaluation

Conclusion

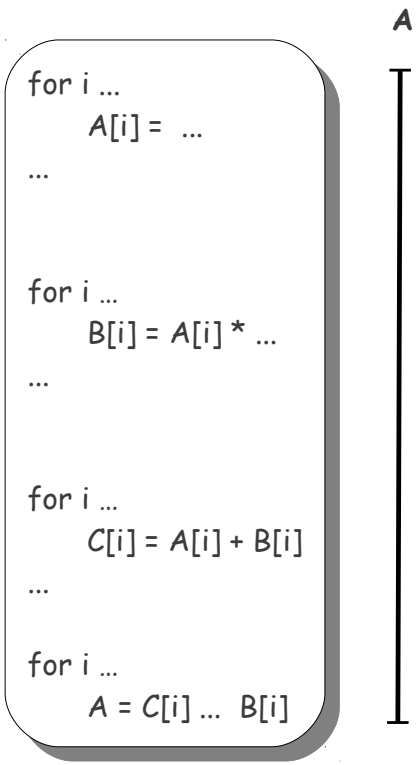
# LM Allocation and Weighted Interval Graph (WIG) Coloring

- Given a numbered intermediate representation of a program
- The live range of the arrays approximated as intervals
- The local memory allocation problem for a linearized program is equivalent to WIG coloring problem called the shipbuilding problem

```
for i ...
    A[i] = ...
...


for i ...
    B[i] = A[i] * ...
...


for i ...
    C[i] = A[i] + B[i]
...

for i ...
    A = C[i] ... B[i]
```

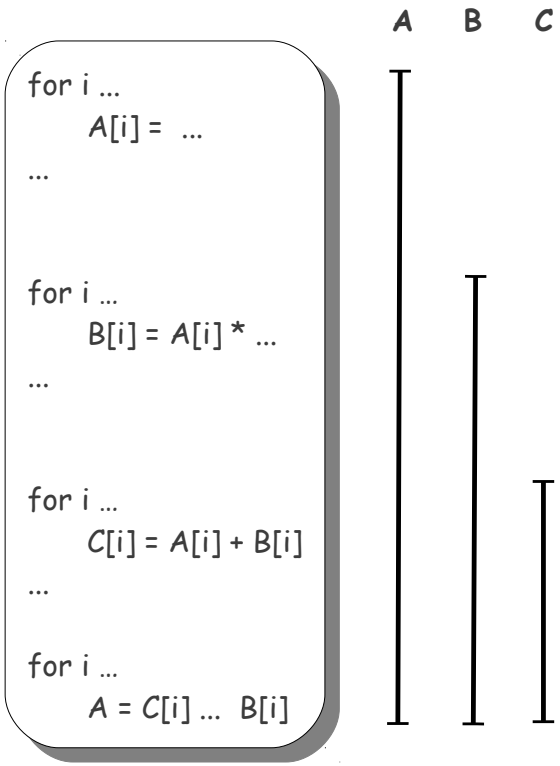
# LM Allocation and Weighted Interval Graph (WIG) Coloring

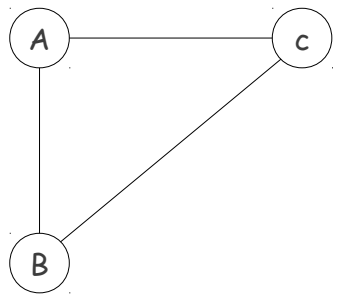
- Given a numbered intermediate representation of a program
- The live range of the arrays approximated as intervals
- The local memory allocation problem for a linearized program is equivalent to WIG coloring problem called the shipbuilding problem



# LM Allocation and Weighted Interval Graph (WIG) Coloring

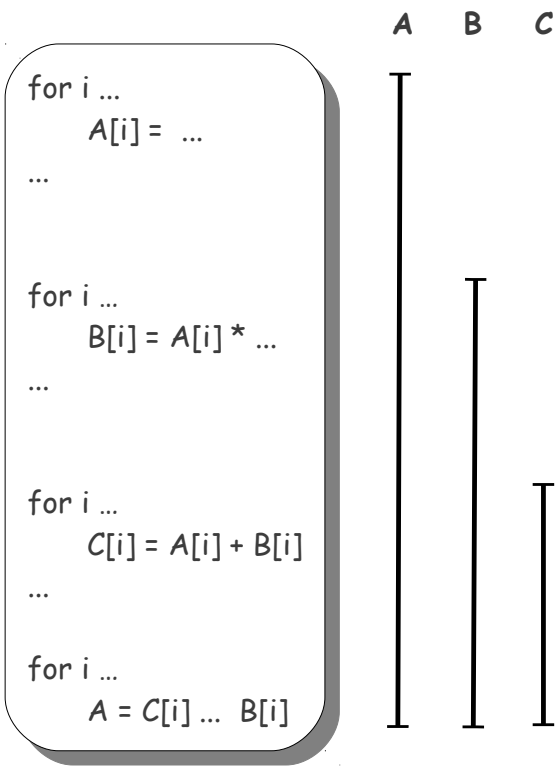
- Given a numbered intermediate representation of a program
- The live range of the arrays approximated as intervals
- The local memory allocation problem for a linearized program is equivalent to WIG coloring problem called the shipbuilding problem

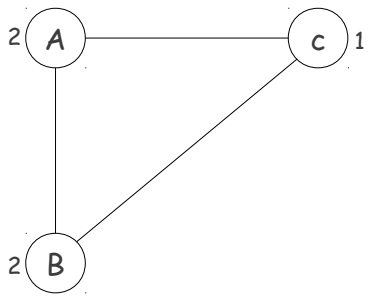


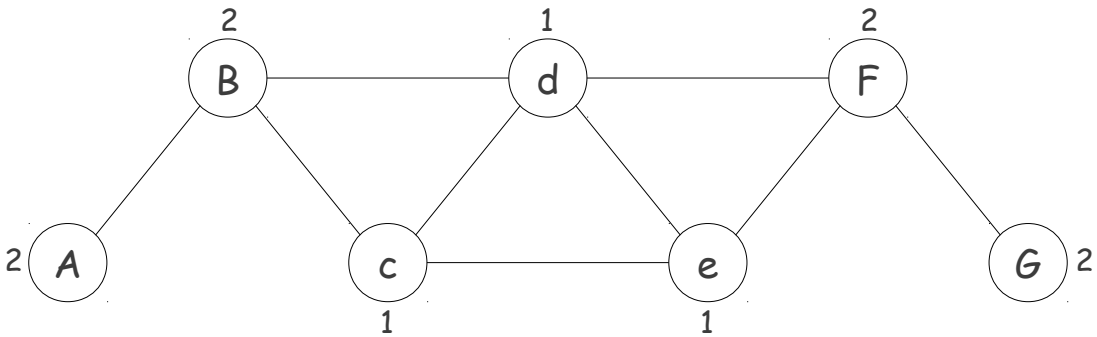
# LM Allocation and Weighted Interval Graph (WIG) Coloring

- Given a numbered intermediate representation of a program
- The live range of the arrays approximated as intervals
- The local memory allocation problem for a linearized program is equivalent to WIG coloring problem called the shipbuilding problem



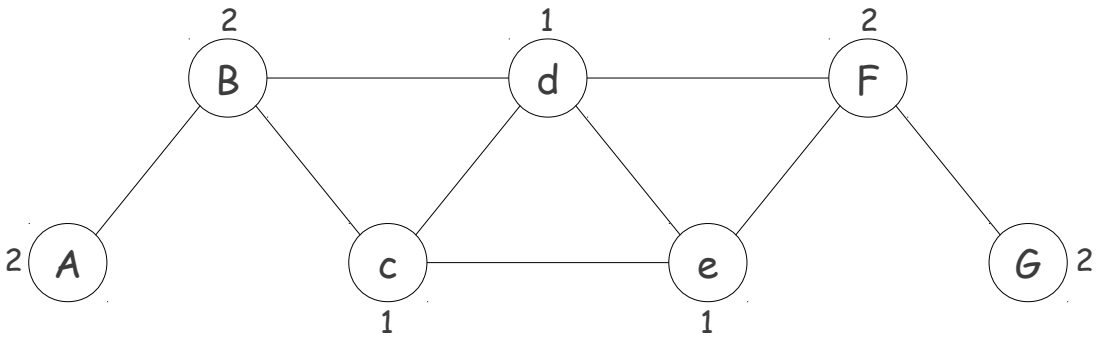



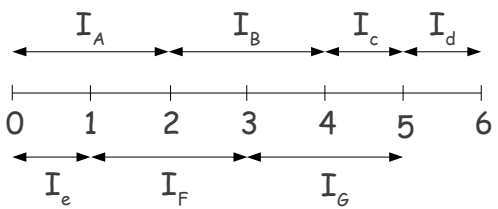
# The Shipbuilding Problem




- Determining whether $\chi(G_w) \leq k$ is NP-complete [Golumbic'04]

# The Shipbuilding Problem

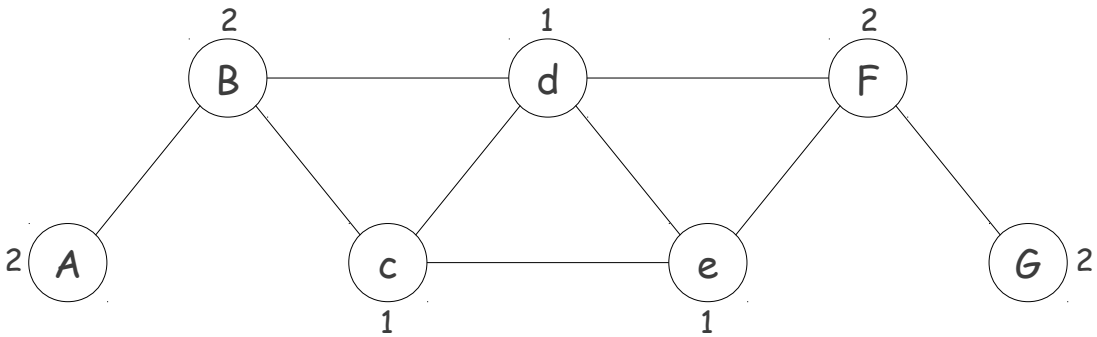


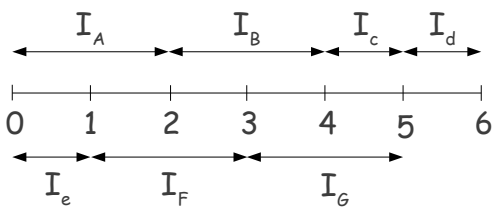




Interval coloring
with **6** colors

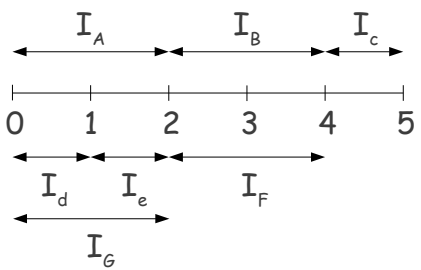
- Determining whether $\chi(G_w) \leq k$ is NP-complete [Golumbic'04]

# The Shipbuilding Problem



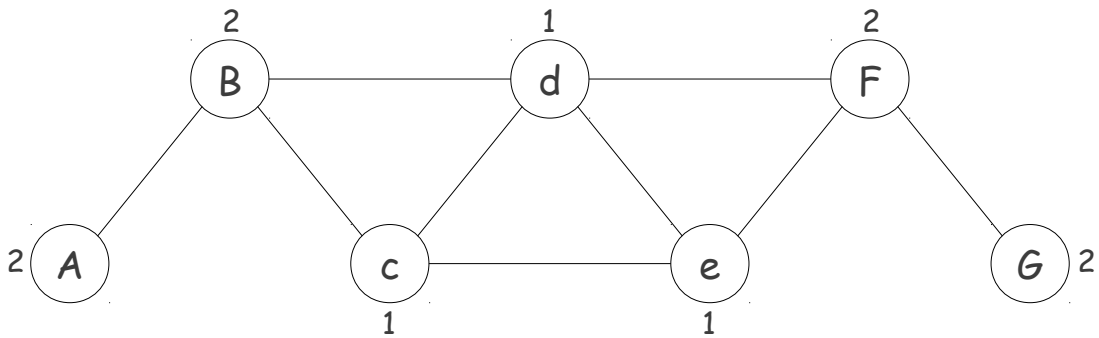

Interval coloring with **6** colors

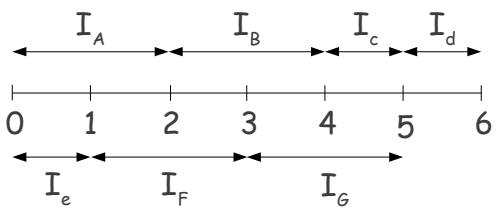



Interval coloring with the chromatic number: **5** colors

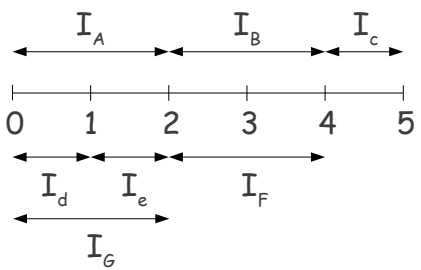
- Determining whether $\chi(G_w) \leq k$ is NP-complete [Golumbic'04]

# The Shipbuilding Problem




Interval coloring with **6** colors

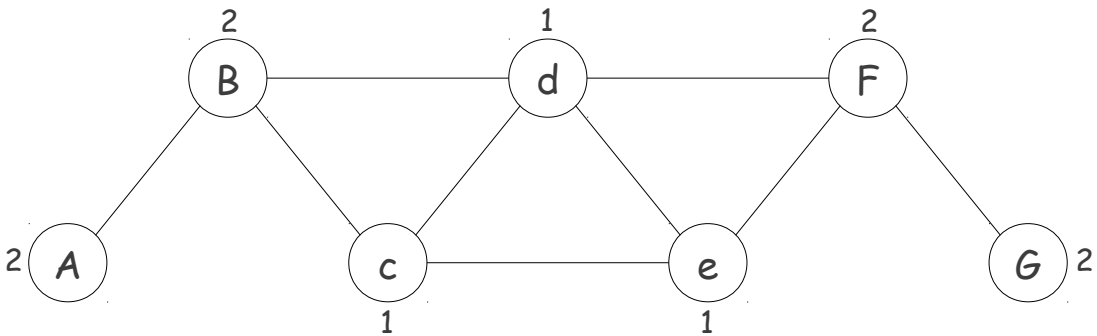


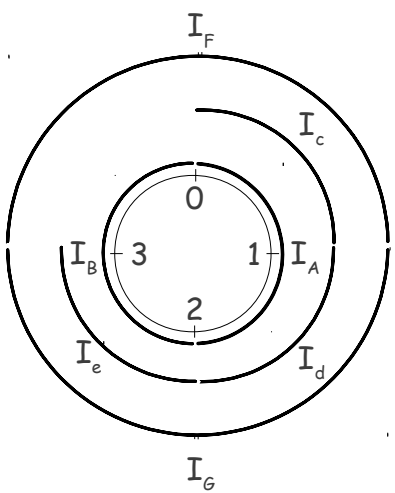

Interval coloring with the chromatic number: **5** colors

- Determining whether $\chi(G_w) \leq k$ is NP-complete [Golumbic'04]

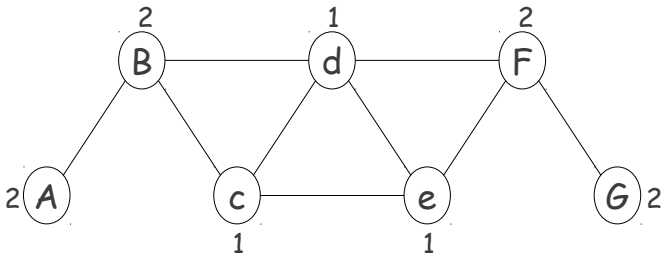
# The Submarine-building Problem

- Assuming that loads and stores wrap around transparently
- We can design a new variant of the shipbuilding problem: the submarine-building problem.
- More flexibility to choose the offsets, i.e., to perform the assignment: e.g. $I_F$



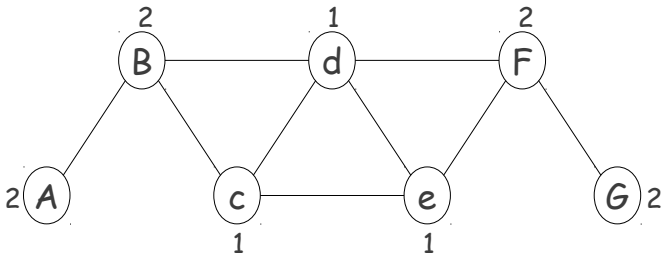



# The Submarine-building Problem is NP-complete on WIG



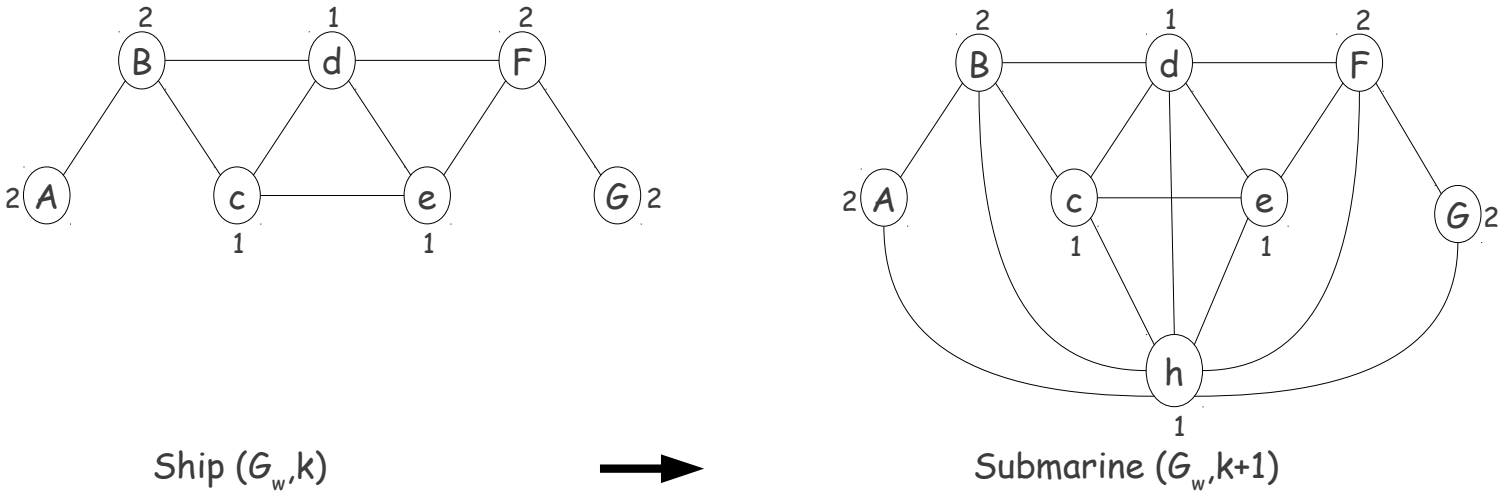

Ship $(G_w, k)$

# The Submarine-building Problem is NP-complete on WIG

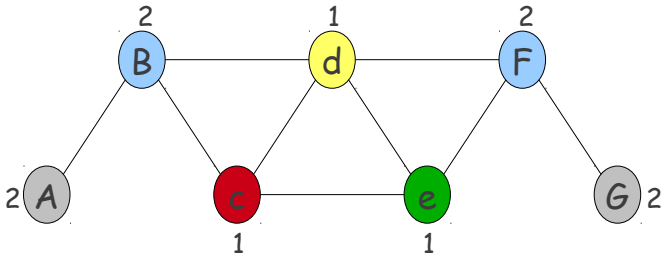


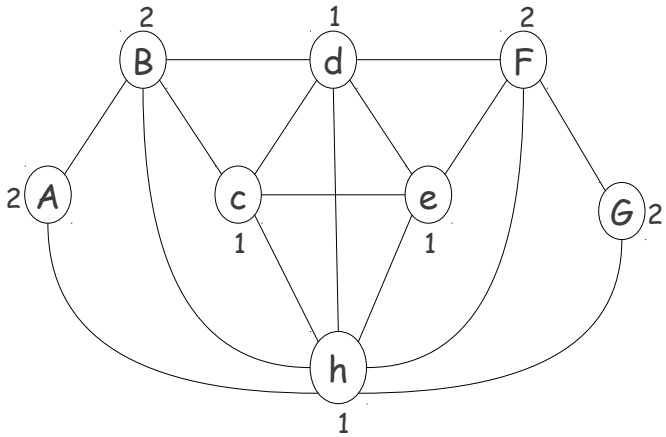

Ship $(G_w, k)$ →

# The Submarine-building Problem is NP-complete on WIG

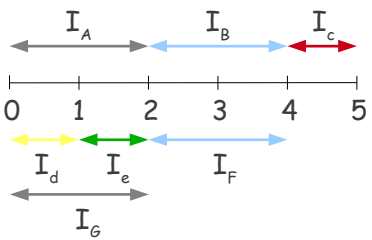


# The Submarine-building Problem is NP-complete on WIG



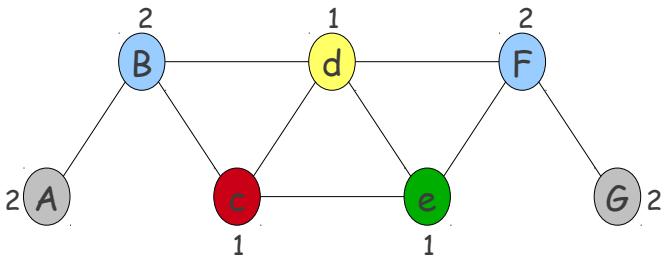



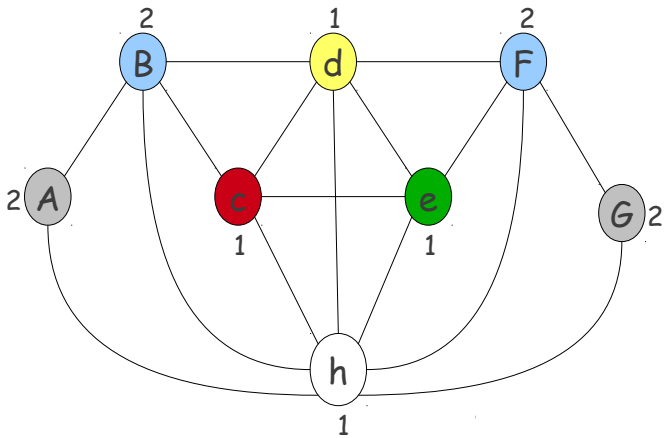

Ship ($G_w$,k) → Submarine ($G_w$,k+1)



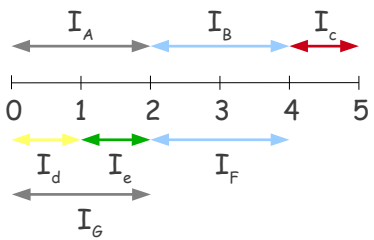

Color intervals

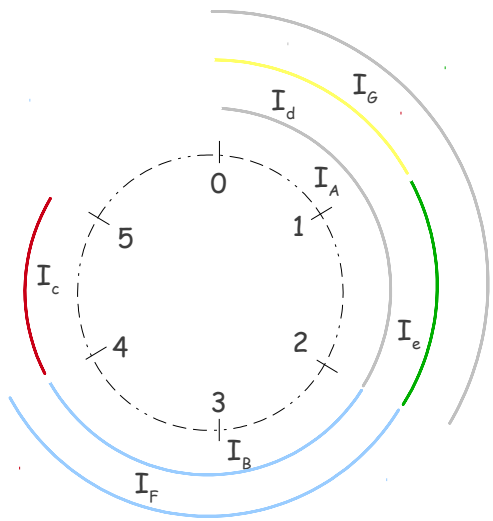
# The Submarine-building Problem is NP-complete on WIG



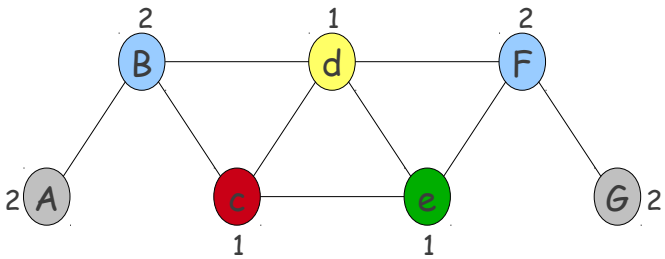

Ship ($G_w$,k) → Submarine ($G_w$,k+1)

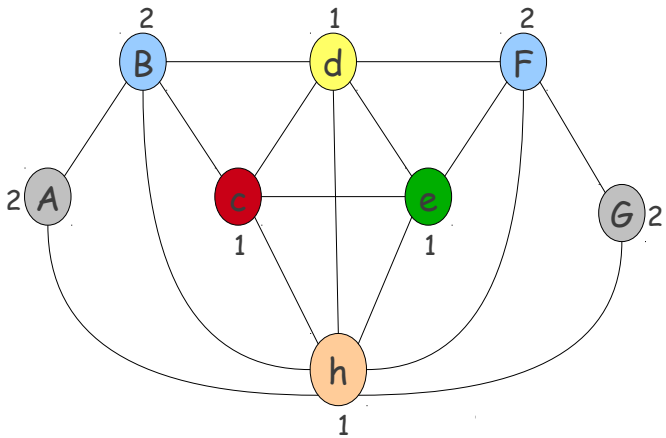


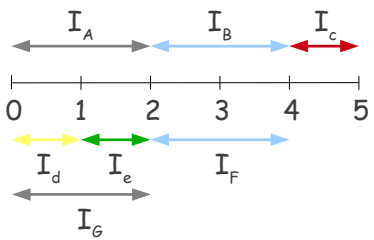

Color intervals → Color intervals

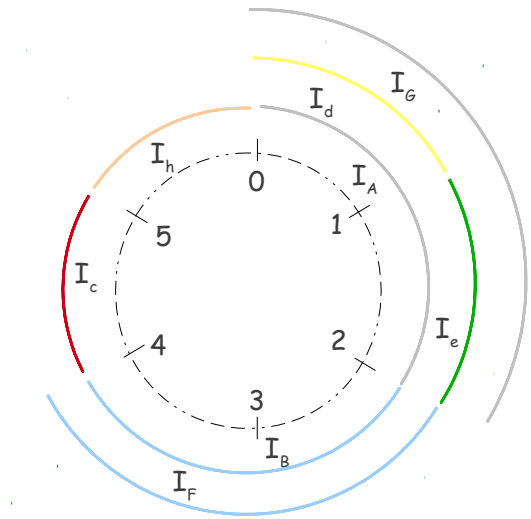
# The Submarine-building Problem is NP-complete on WIG



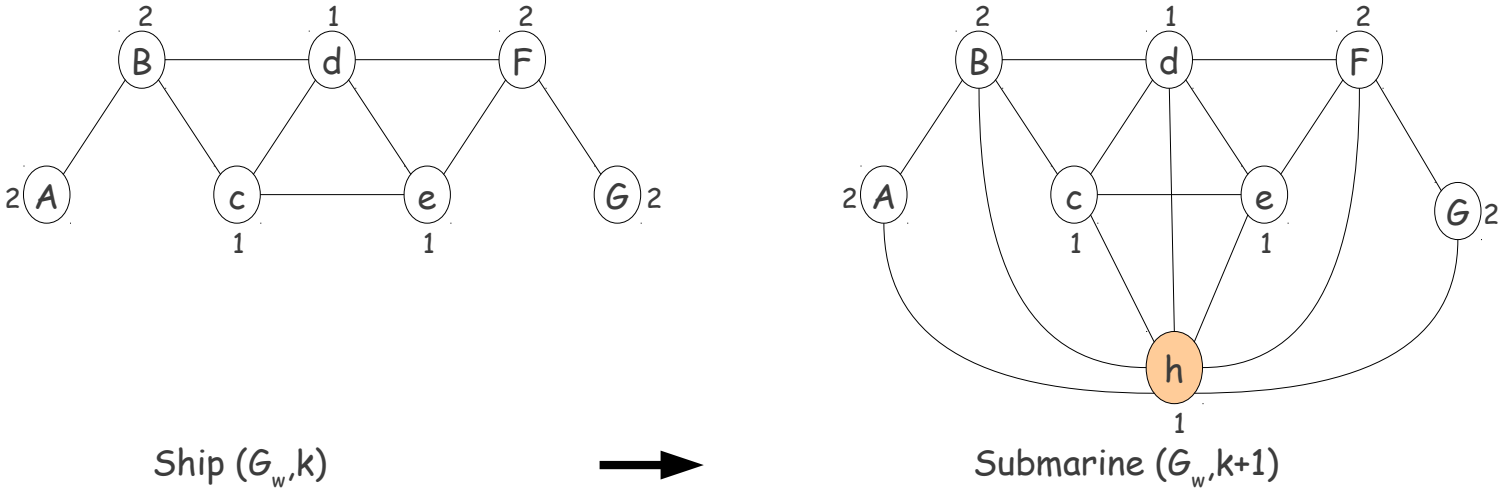

Ship ($G_w$,k) → Submarine ($G_w$,k+1)

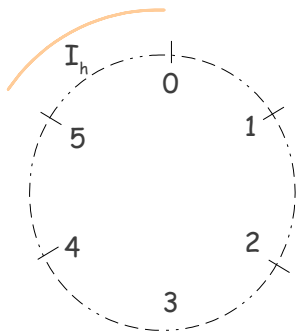



Color intervals → Color intervals

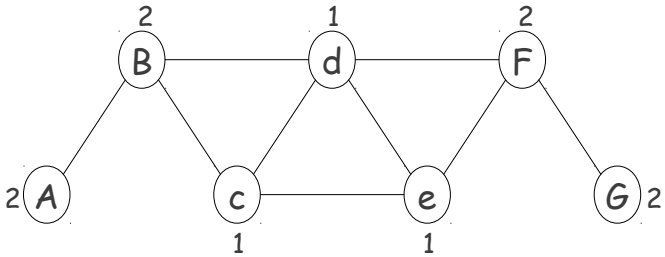
# The Submarine-building Problem is NP-complete on WIG

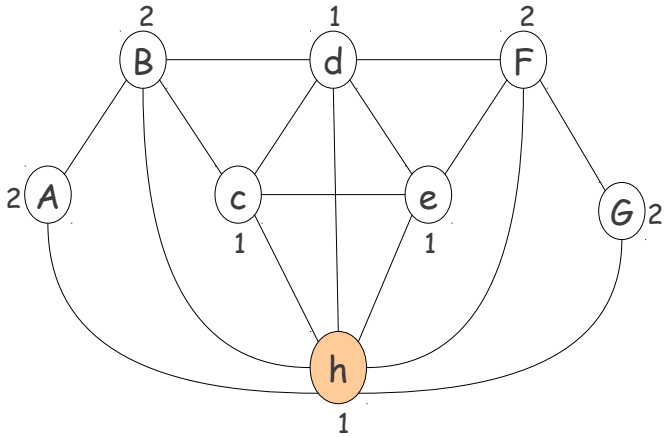


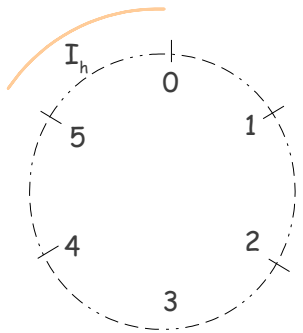

Color intervals

# The Submarine-building Problem is NP-complete on WIG



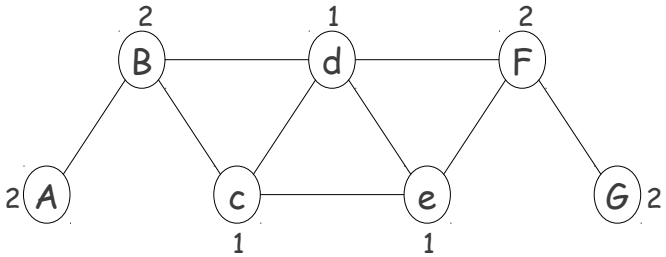



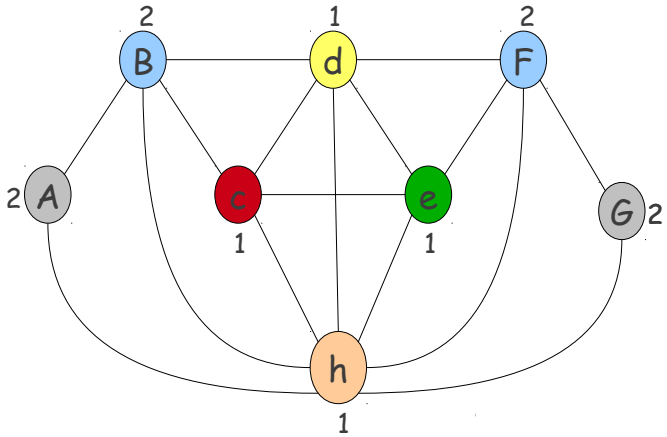

Ship ($G_w$,k) → Submarine ($G_w$,k+1)

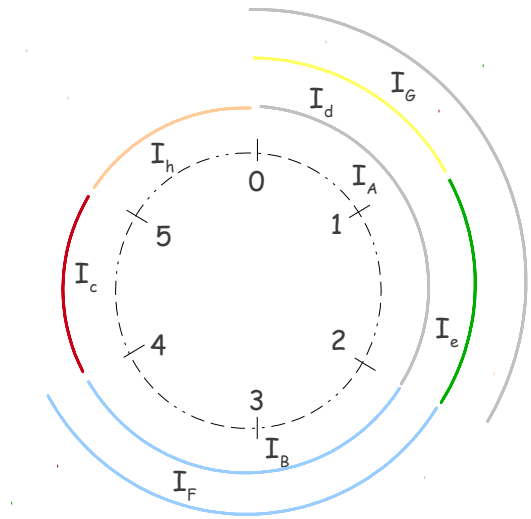

Color intervals ← Color intervals

# The Submarine-building Problem is NP-complete on WIG



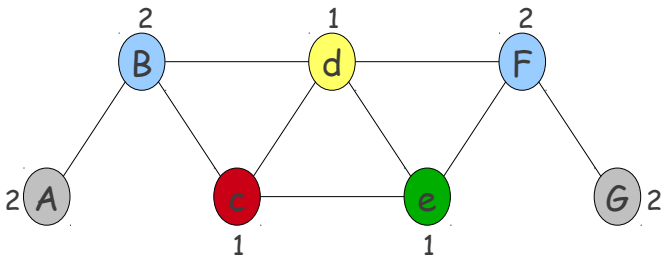



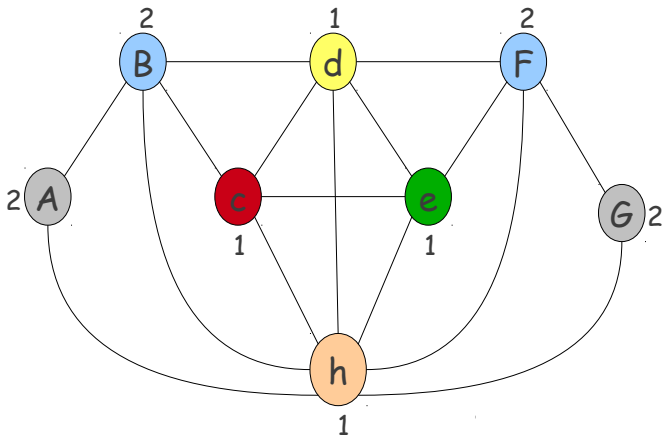

Ship ($G_w$,k) → Submarine ($G_w$,k+1)

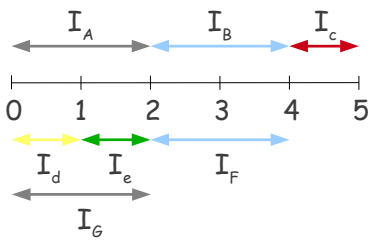

Color intervals ← Color intervals

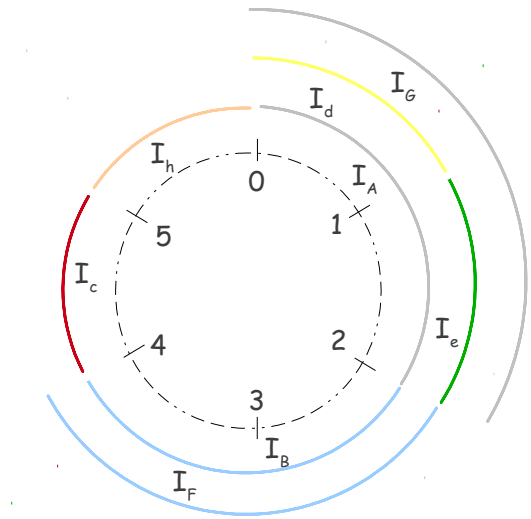
# The Submarine-building Problem is NP-complete on WIG



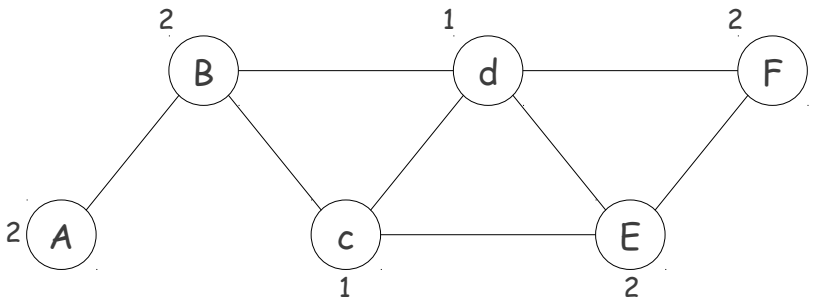

Ship ($G_w$,k) → Submarine ($G_w$,k+1)



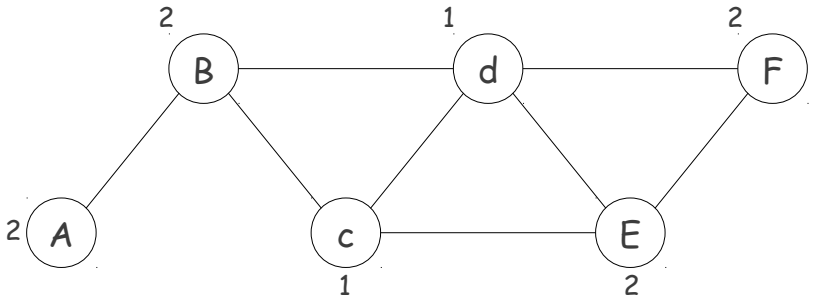



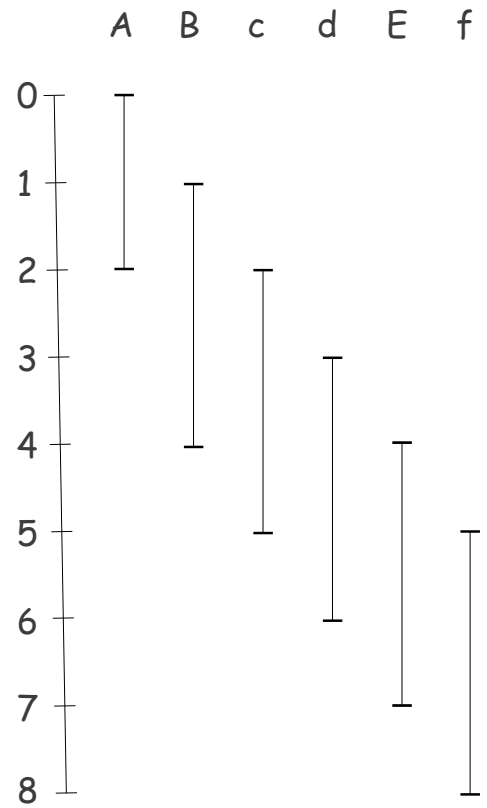

Color intervals ← Color intervals

# The Submarine-building Problem is Linear on Proper Interval Graphs

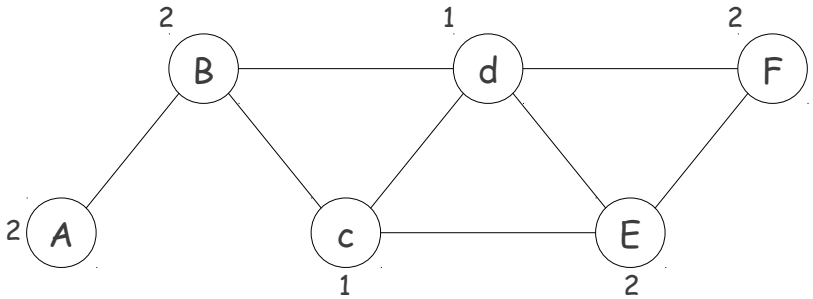


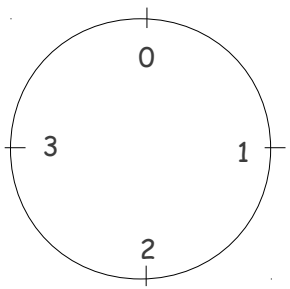
# The Submarine-building Problem is Linear on Proper Interval Graphs



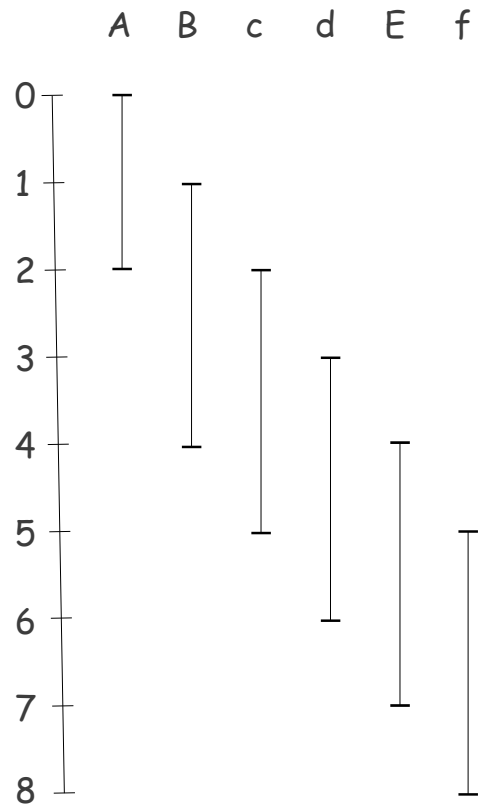



# The Submarine-building Problem is Linear on Proper Interval Graphs



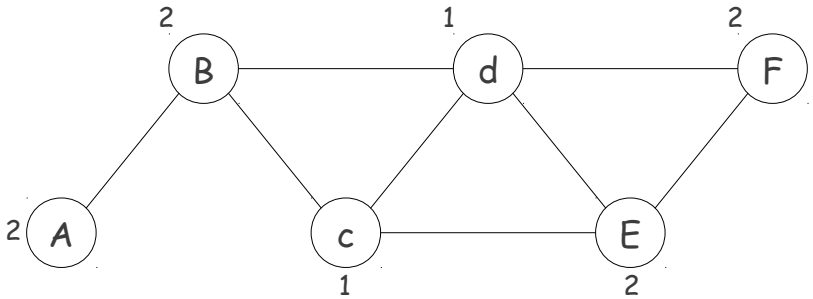



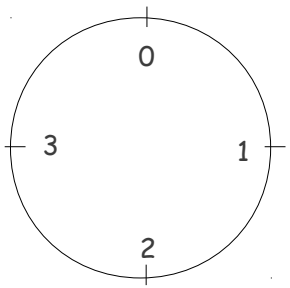

Color intervals



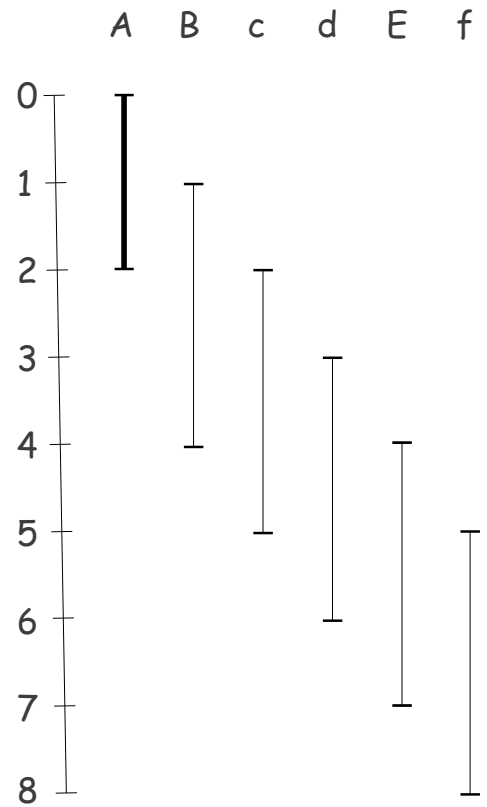

Program points

# The Submarine-building Problem is Linear on Proper Interval Graphs



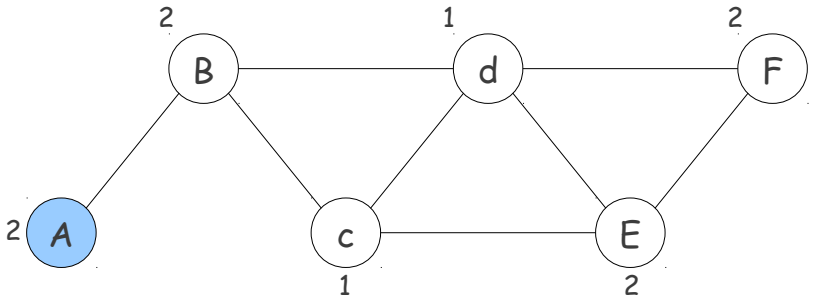



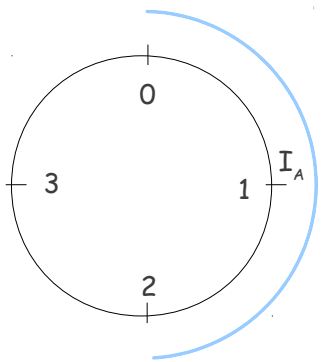

Color intervals



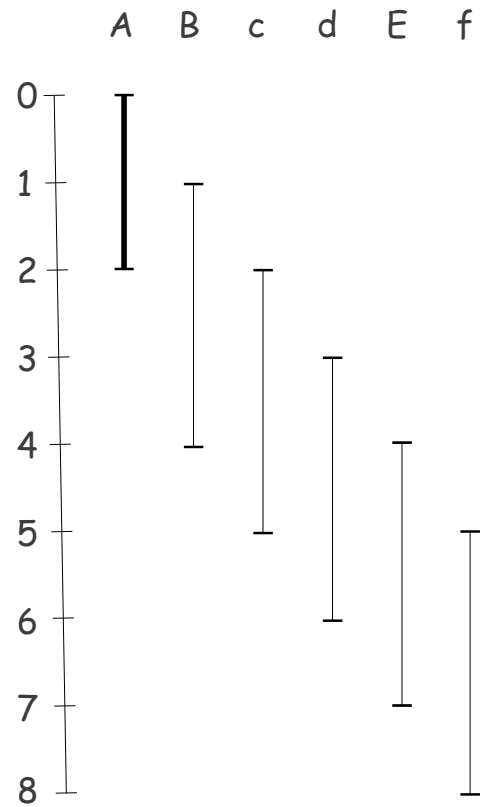

Program points

# The Submarine-building Problem is Linear on Proper Interval Graphs



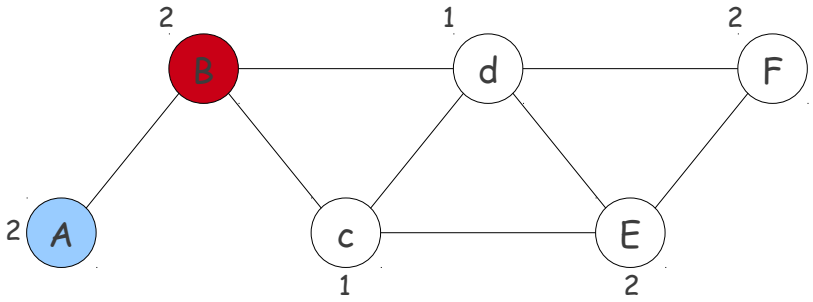



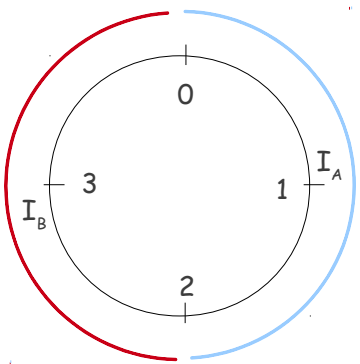

Color intervals



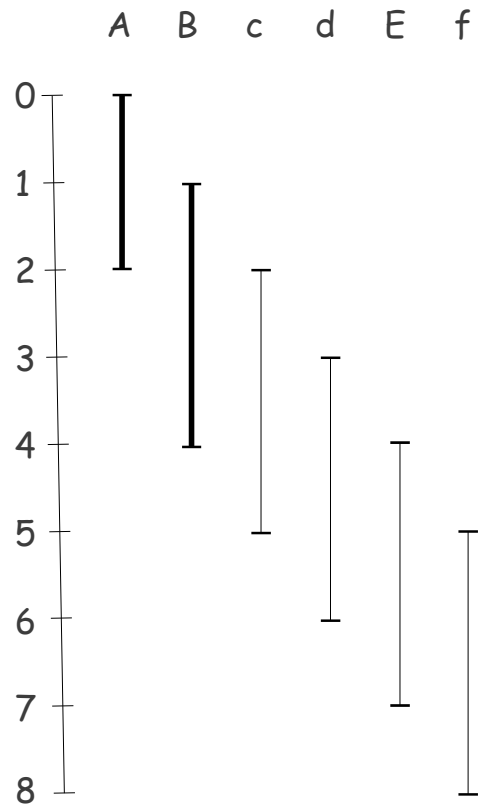

Program points

# The Submarine-building Problem is Linear on Proper Interval Graphs



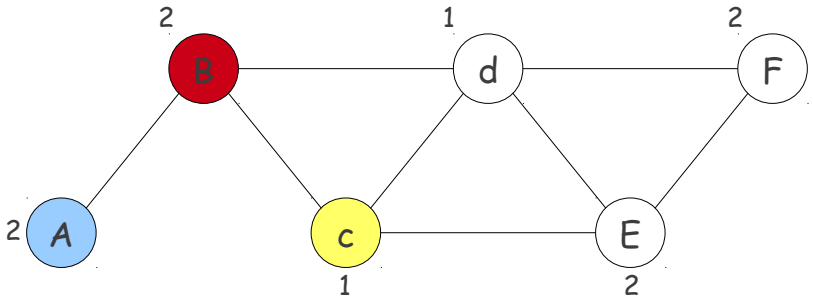



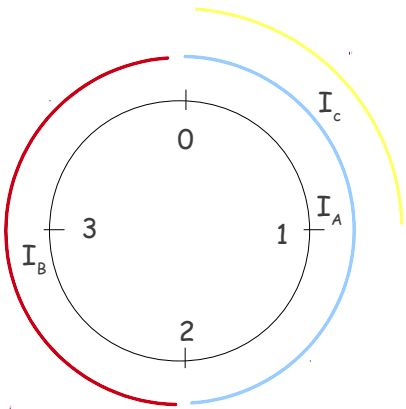

Color intervals



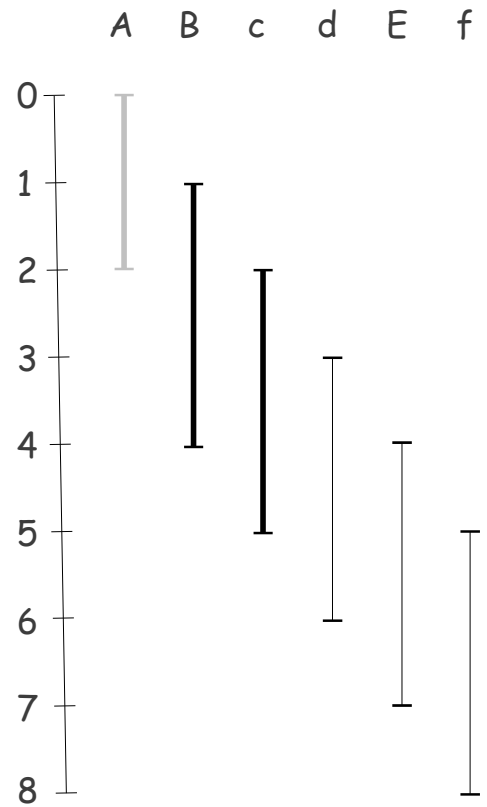

Program points

# The Submarine-building Problem is Linear on Proper Interval Graphs



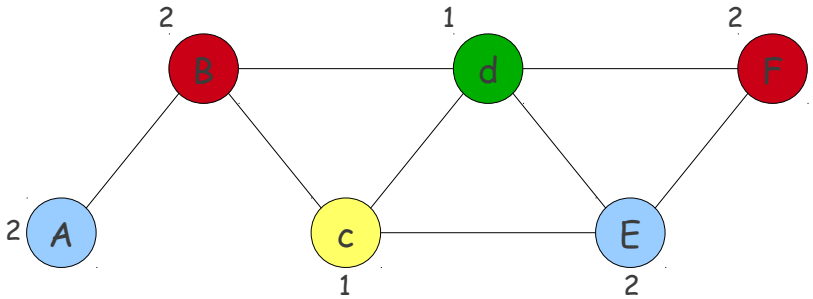



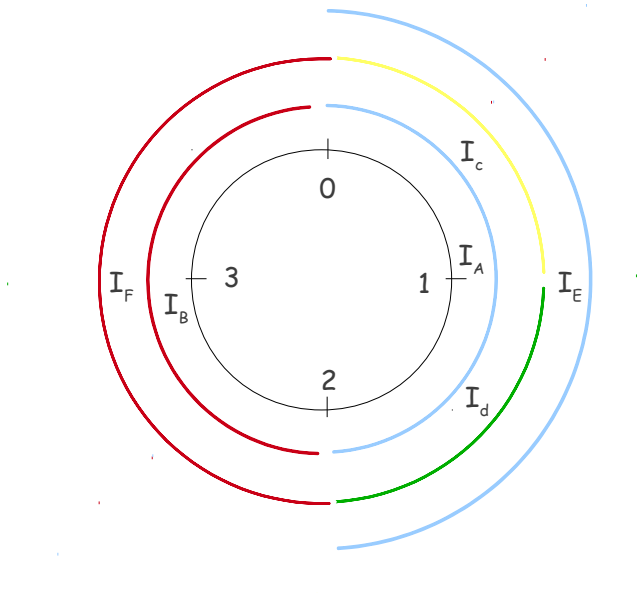

Color intervals



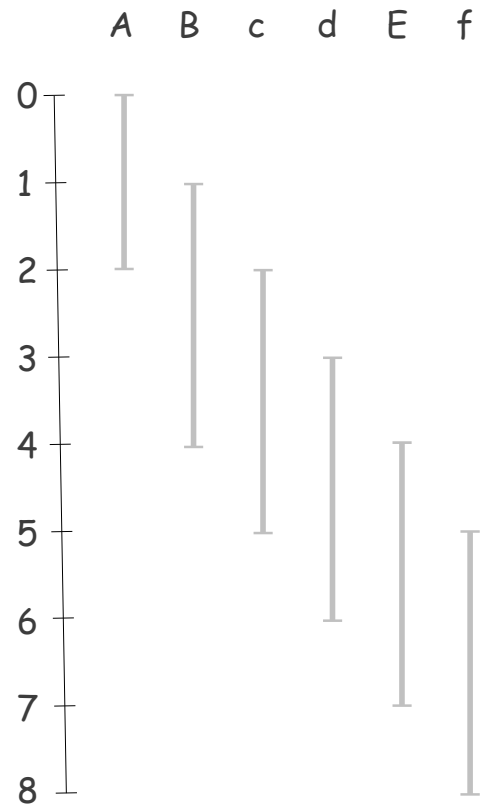

Program points

# The Submarine-building Problem is Linear on Proper Interval Graphs



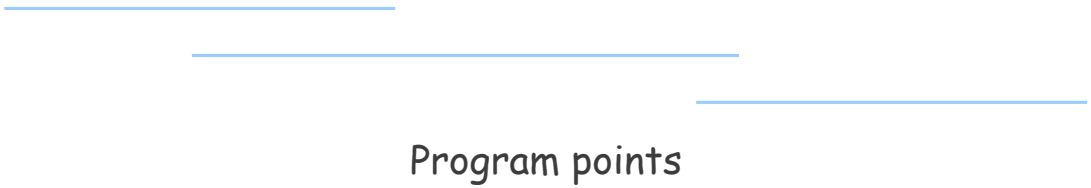




Color intervals



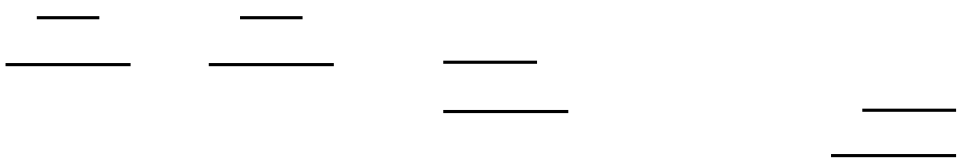

Program points

# Not-So-Proper (NSP) Weighted Interval Graphs



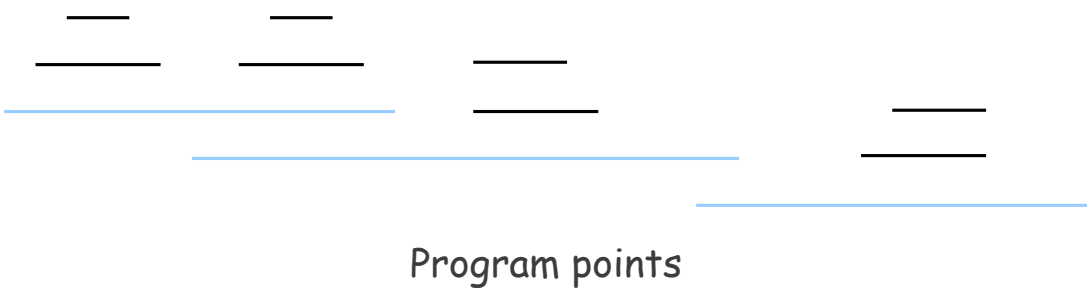

- The submarine-building problem is linear on the class of proper interval graphs whereas the shipbuilding problem remains NP-complete on proper interval graphs
- The submarine-building problem is also linear on the class of superperfect graphs observed in many embedded application [Li'11]
- The class of NSP interval graphs, that generalizes both proper and superperfect graphs, are used to decouple the local memory allocation

# Not-So-Proper (NSP) Weighted Interval Graphs




- The submarine-building problem is linear on the class of proper interval graphs whereas the shipbuilding problem remains NP-complete on proper interval graphs
- The submarine-building problem is also linear on the class of superperfect graphs observed in many embedded application [Li'11]
- The class of NSP interval graphs, that generalizes both proper and superperfect graphs, are used to decouple the local memory allocation

# Not-So-Proper (NSP) Weighted Interval Graphs




- The submarine-building problem is linear on the class of proper interval graphs whereas the shipbuilding problem remains NP-complete on proper interval graphs
- The submarine-building problem is also linear on the class of superperfect graphs observed in many embedded application [Li'11]
- The class of NSP interval graphs, that generalizes both proper and superperfect graphs, are used to decouple the local memory allocation

# Outline

Introduction

Weighted Interval Graph Coloring

LM Allocation through NSP WIG Coloring

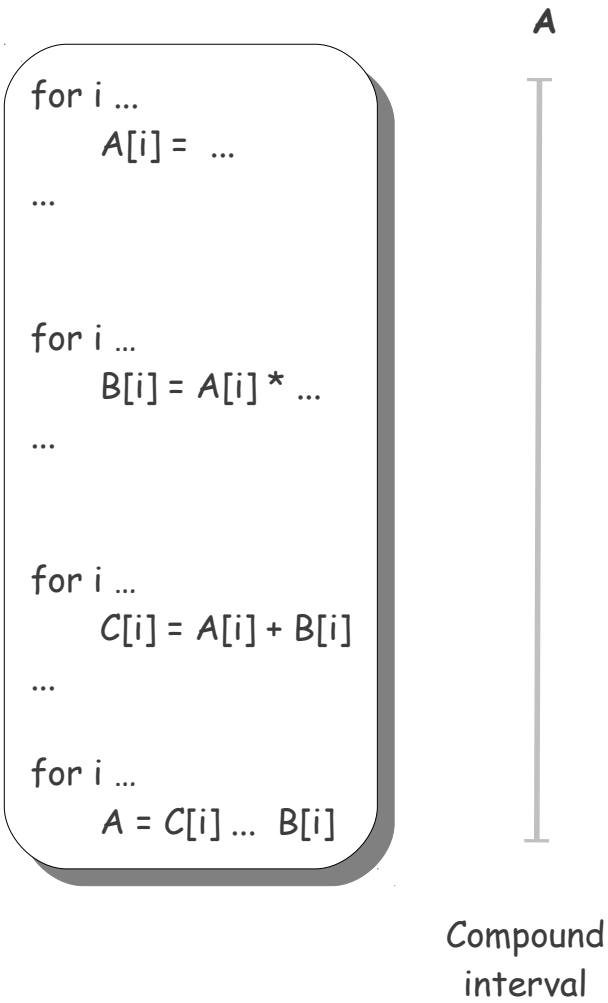
Experimental Evaluation

Conclusion

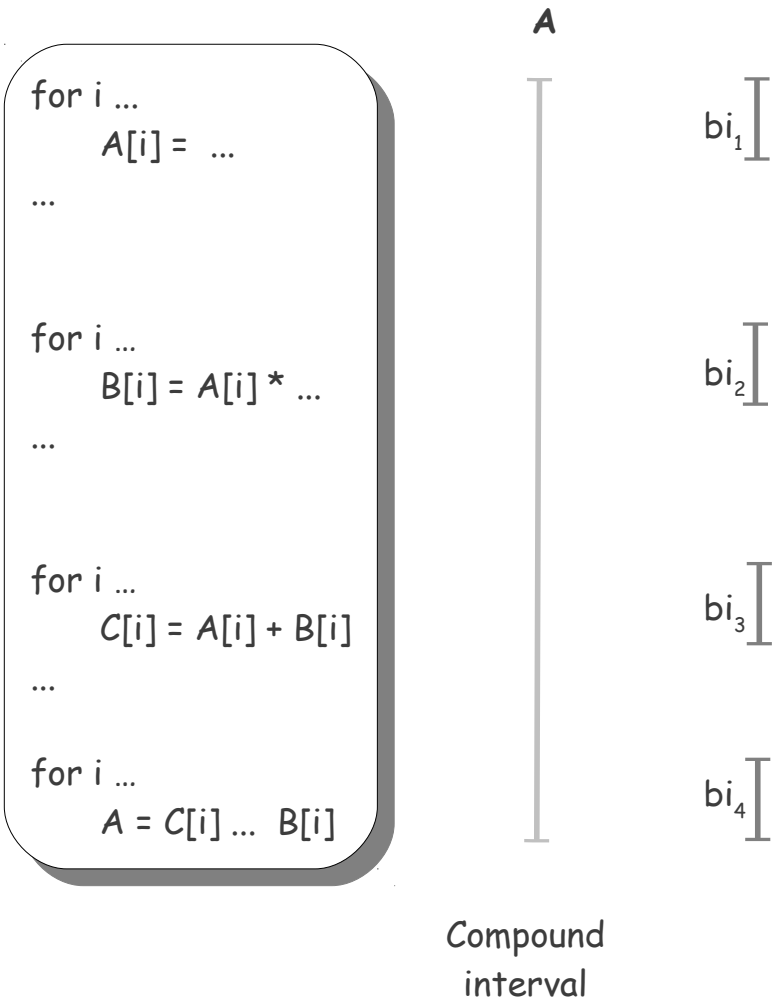
# Live Range Representations

```
for i ...
    A[i] =  ...
...


for i ...
    B[i] = A[i] * ...
...


for i ...
    C[i] = A[i] + B[i]
...

for i ...
    A = C[i] ...  B[i]
```

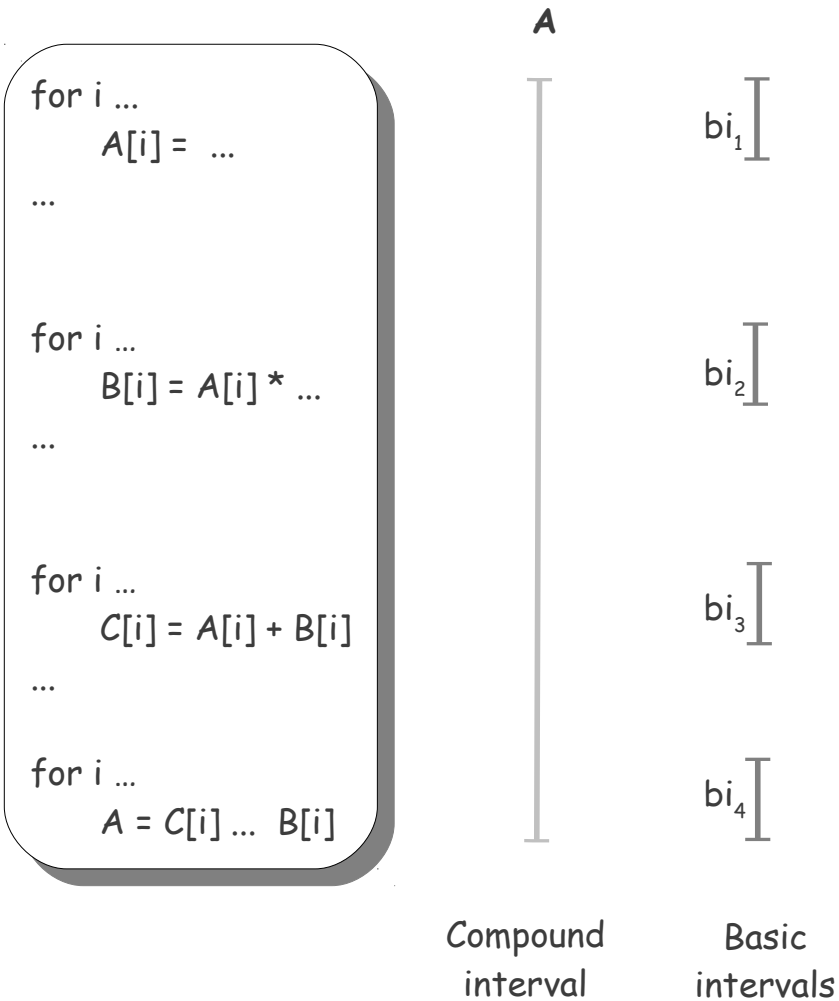
# Live Range Representations



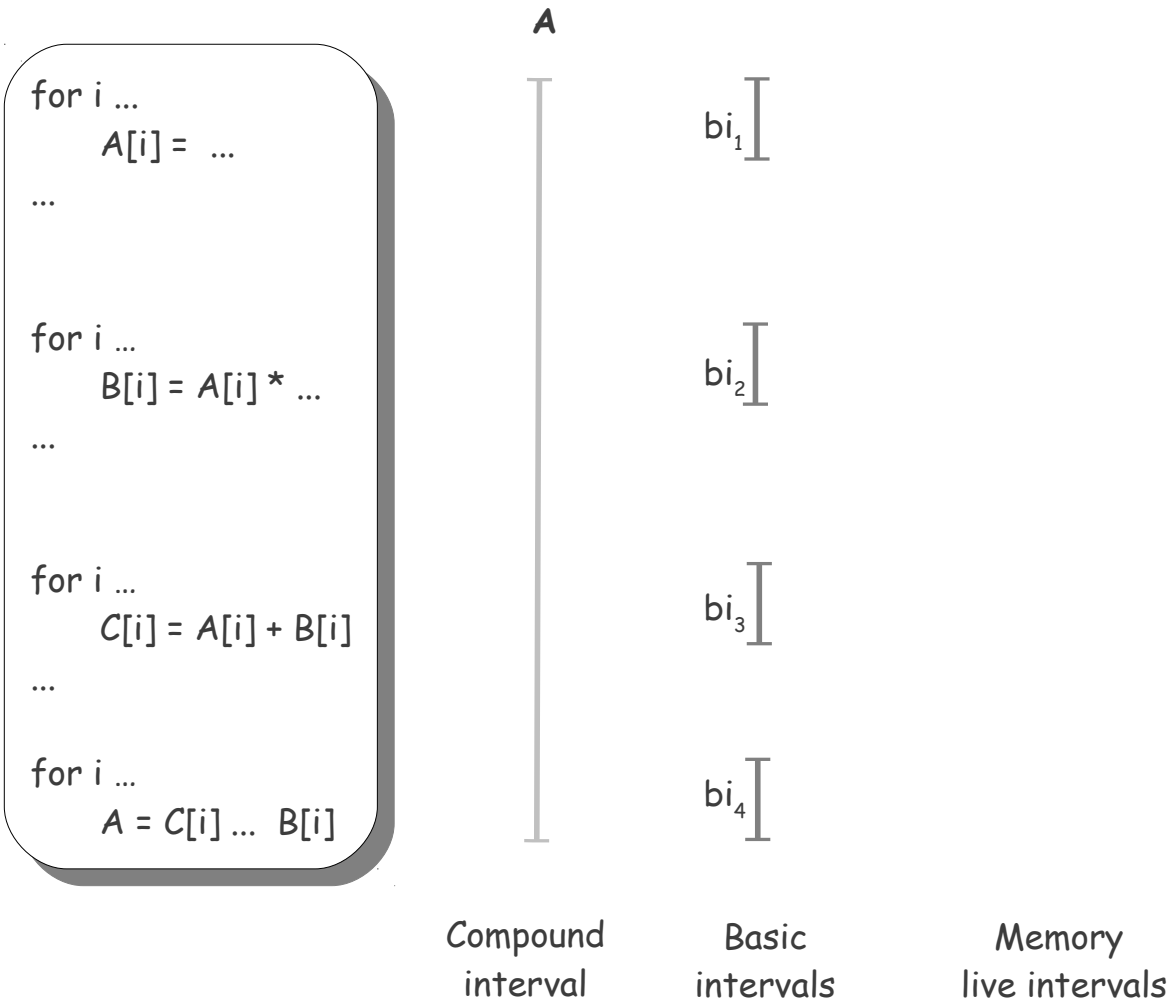
# Live Range Representations



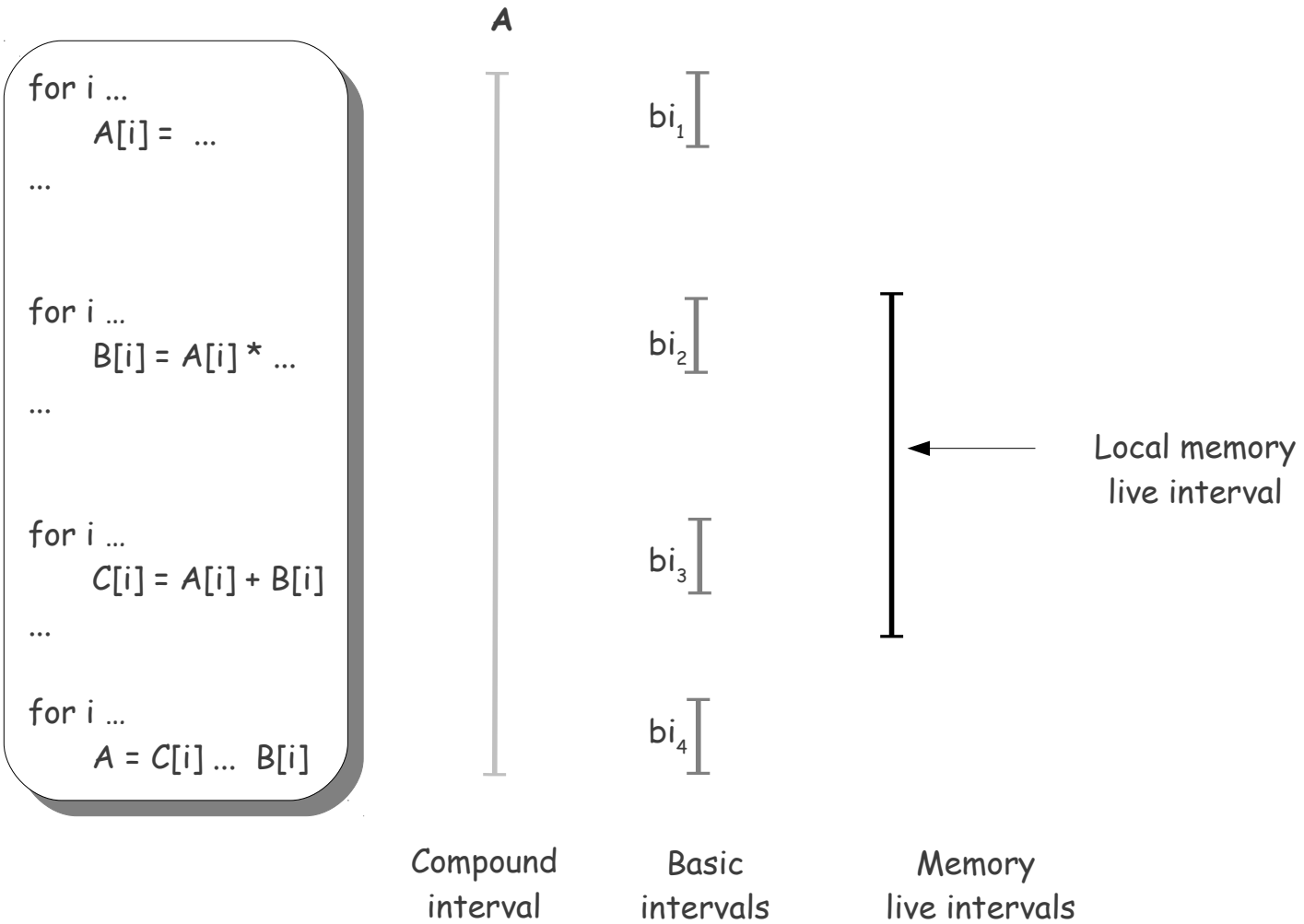
# Live Range Representations



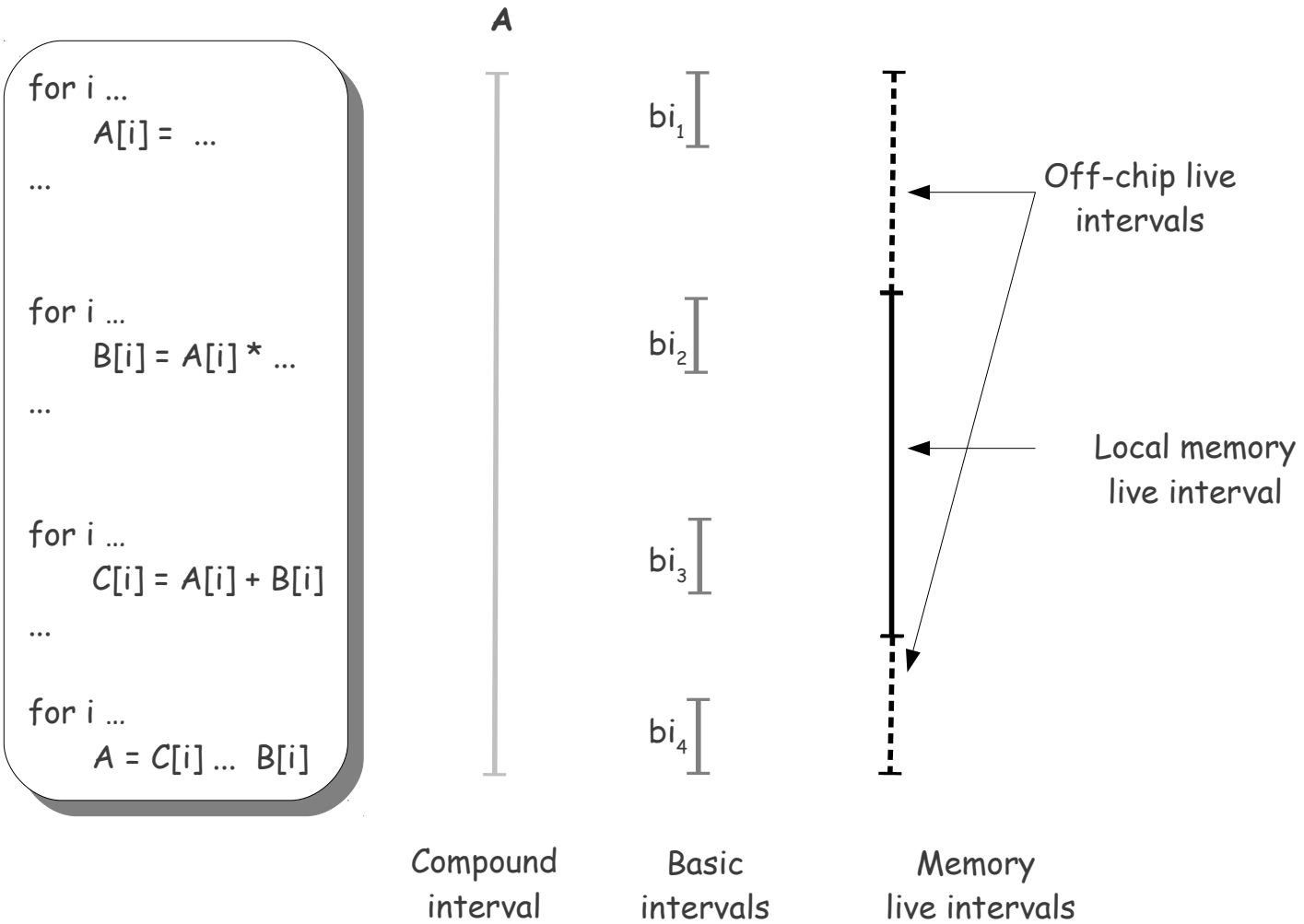
# Live Range Representations



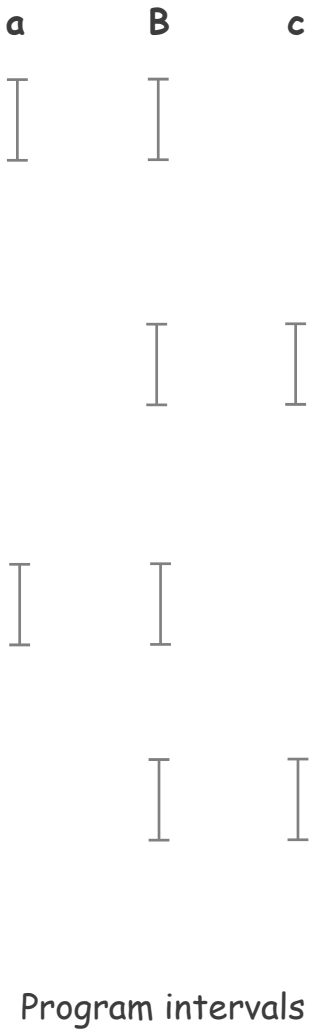
# Live Range Representations



# Live Range Representations



# Approximating a WIG to a NSP WIG

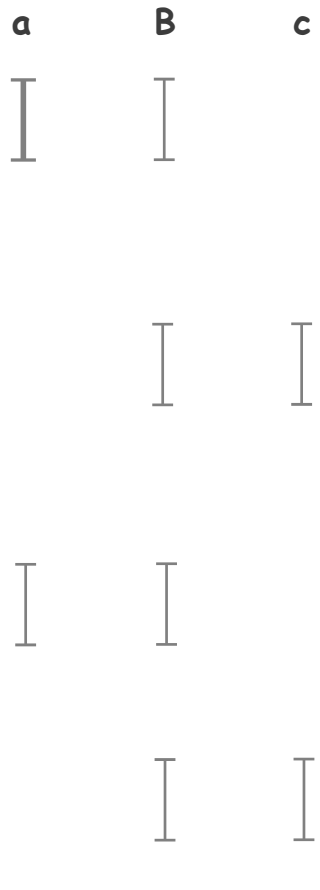



Walk over the list of basic intervals and create a NSP WIG $G_w$ composed of local memory intervals

For each basic interval `bi`

- If there is an existing local memory interval that can receive `bi` then add `bi` to the local memory interval
- Else If a freshly local memory interval containing only `bi` can be added to the $G_w$ without breaking the NSP property or making maxSize greater than the LM size then add it to $G_w$
- Else either make of `bi` an off-chip live range or spill or split some local memory candidates to make room for `bi`

# Approximating a WIG to a NSP WIG



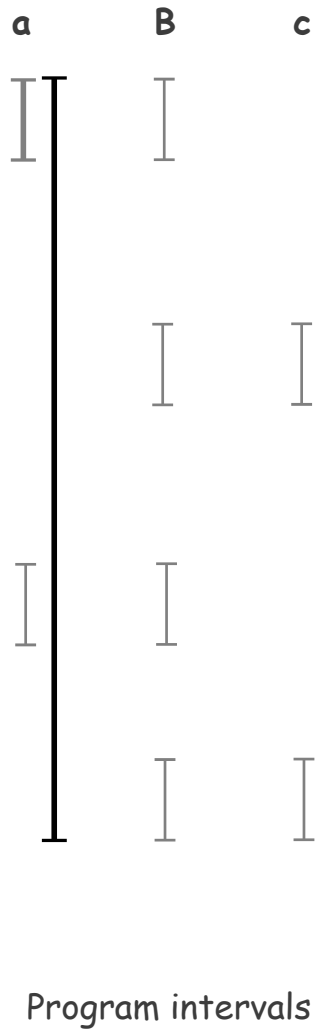

Program intervals

Walk over the list of basic intervals and create a NSP WIG $G_w$ composed of local memory intervals

For each basic interval `bi`

- If there is an existing local memory interval that can receive `bi` then add `bi` to the local memory interval
- Else If a freshly local memory interval containing only `bi` can be added to the $G_w$ without breaking the NSP property or making maxSize greater than the LM size then add it to $G_w$
- Else either make of `bi` an off-chip live range or spill or split some local memory candidates to make room for `bi`

# Approximating a WIG to a NSP WIG



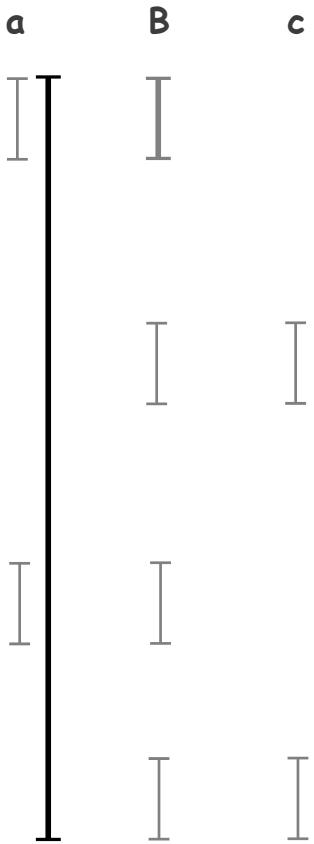

Program intervals

Walk over the list of basic intervals and create a NSP WIG $G_w$ composed of local memory intervals

For each basic interval `bi`

- If there is an existing local memory interval that can receive `bi` then add `bi` to the local memory interval
- Else If a freshly local memory interval containing only `bi` can be added to the $G_w$ without breaking the NSP property or making maxSize greater than the LM size then add it to $G_w$
- Else either make of `bi` an off-chip live range or spill or split some local memory candidates to make room for `bi`

# Approximating a WIG to a NSP WIG



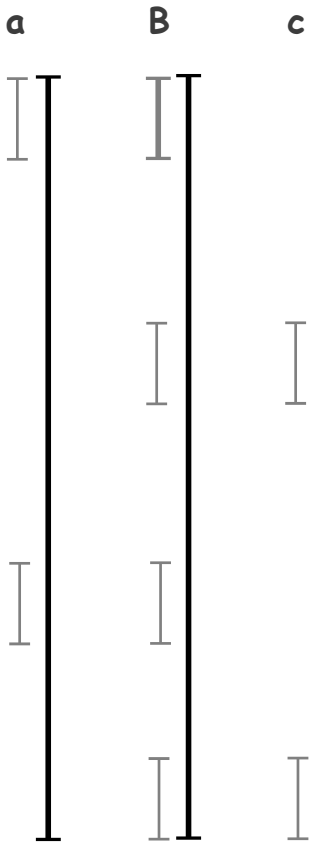

Program intervals

Walk over the list of basic intervals and create a NSP WIG $G_w$ composed of local memory intervals

For each basic interval `bi`

- If there is an existing local memory interval that can receive `bi` then add `bi` to the local memory interval
- Else If a freshly local memory interval containing only `bi` can be added to the $G_w$ without breaking the NSP property or making maxSize greater than the LM size then add it to $G_w$
- Else either make of `bi` an off-chip live range or spill or split some local memory candidates to make room for `bi`

# Approximating a WIG to a NSP WIG



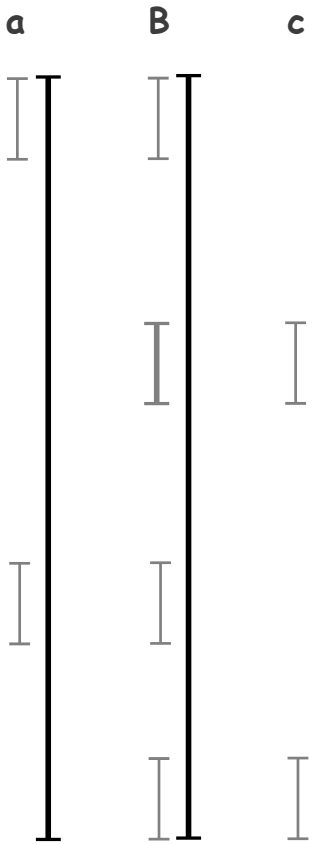

Program intervals

Walk over the list of basic intervals and create a NSP WIG $G_w$ composed of local memory intervals

For each basic interval `bi`

- If there is an existing local memory interval that can receive `bi` then add `bi` to the local memory interval
- Else If a freshly local memory interval containing only `bi` can be added to the $G_w$ without breaking the NSP property or making maxSize greater than the LM size then add it to $G_w$
- Else either make of `bi` an off-chip live range or spill or split some local memory candidates to make room for `bi`

# Approximating a WIG to a NSP WIG



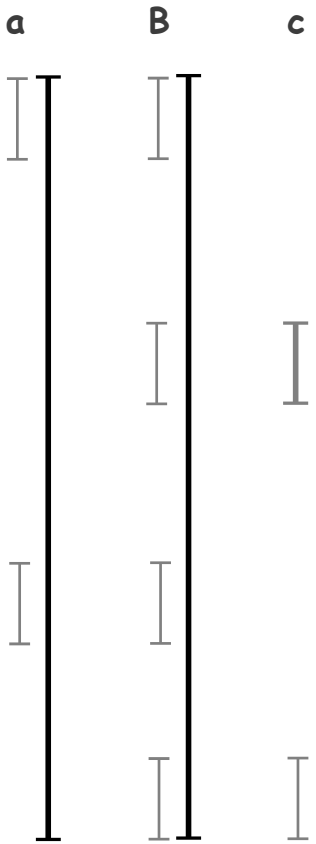

Program intervals

Walk over the list of basic intervals and create a NSP WIG $G_w$ composed of local memory intervals

For each basic interval `bi`

- If there is an existing local memory interval that can receive `bi` then add `bi` to the local memory interval
- Else If a freshly local memory interval containing only `bi` can be added to the $G_w$ without breaking the NSP property or making maxSize greater than the LM size then add it to $G_w$
- Else either make of `bi` an off-chip live range or spill or split some local memory candidates to make room for `bi`

# Approximating a WIG to a NSP WIG



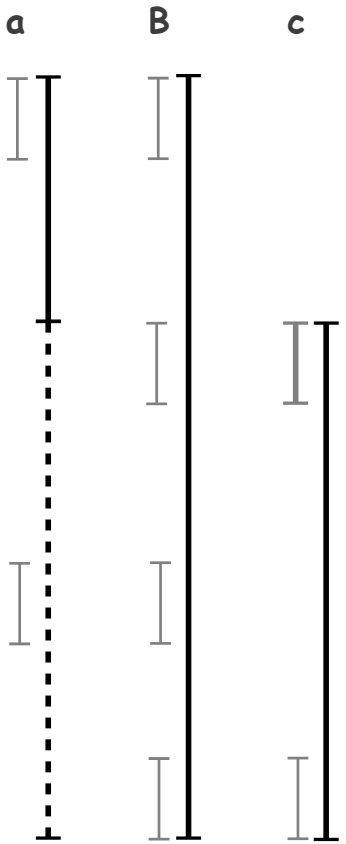

Program intervals

Walk over the list of basic intervals and create a NSP WIG $G_w$ composed of local memory intervals

For each basic interval `bi`

- If there is an existing local memory interval that can receive `bi` then add `bi` to the local memory interval
- Else If a freshly local memory interval containing only `bi` can be added to the $G_w$ without breaking the NSP property or making maxSize greater than the LM size then add it to $G_w$
- Else either make of `bi` an off-chip live range or spill or split some local memory candidates to make room for `bi`

# Approximating a WIG to a NSP WIG



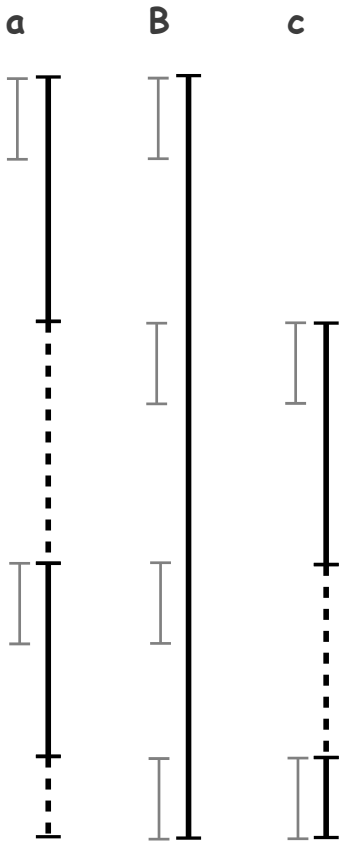

Program intervals

Walk over the list of basic intervals and create a NSP WIG $G_w$ composed of local memory intervals

For each basic interval `bi`

- If there is an existing local memory interval that can receive `bi` then add `bi` to the local memory interval
- Else If a freshly local memory interval containing only `bi` can be added to the $G_w$ without breaking the NSP property or making maxSize greater than the LM size then add it to $G_w$
- Else either make of `bi` an off-chip live range or spill or split some local memory candidates to make room for `bi`

# Approximating a WIG to a NSP WIG




Program intervals

Walk over the list of basic intervals and create a NSP WIG $G_w$ composed of local memory intervals

For each basic interval `bi`

- If there is an existing local memory interval that can receive `bi` then add `bi` to the local memory interval
- Else If a freshly local memory interval containing only `bi` can be added to the $G_w$ without breaking the NSP property or making maxSize greater than the LM size then add it to $G_w$
- Else either make of `bi` an off-chip live range or spill or split some local memory candidates to make room for `bi`

# Outline

Introduction

Weighted Interval Graph Coloring

LM Allocation through NSP WIG Coloring

Experimental Evaluation

Conclusion

# Methodology

## Model Parameters

| Constant | Latency |
|---|---|
| *latency_local_memory* | 8 |
| *latency_main_memory* | 128 |
| *latency_move*($s_v$) | $8 + 2s_v$ |
| *latency_spill*($s_v$) | $128 + 4s_v$ |
| *latency_reload*($s_v$) | $128 + 4s_v$ |

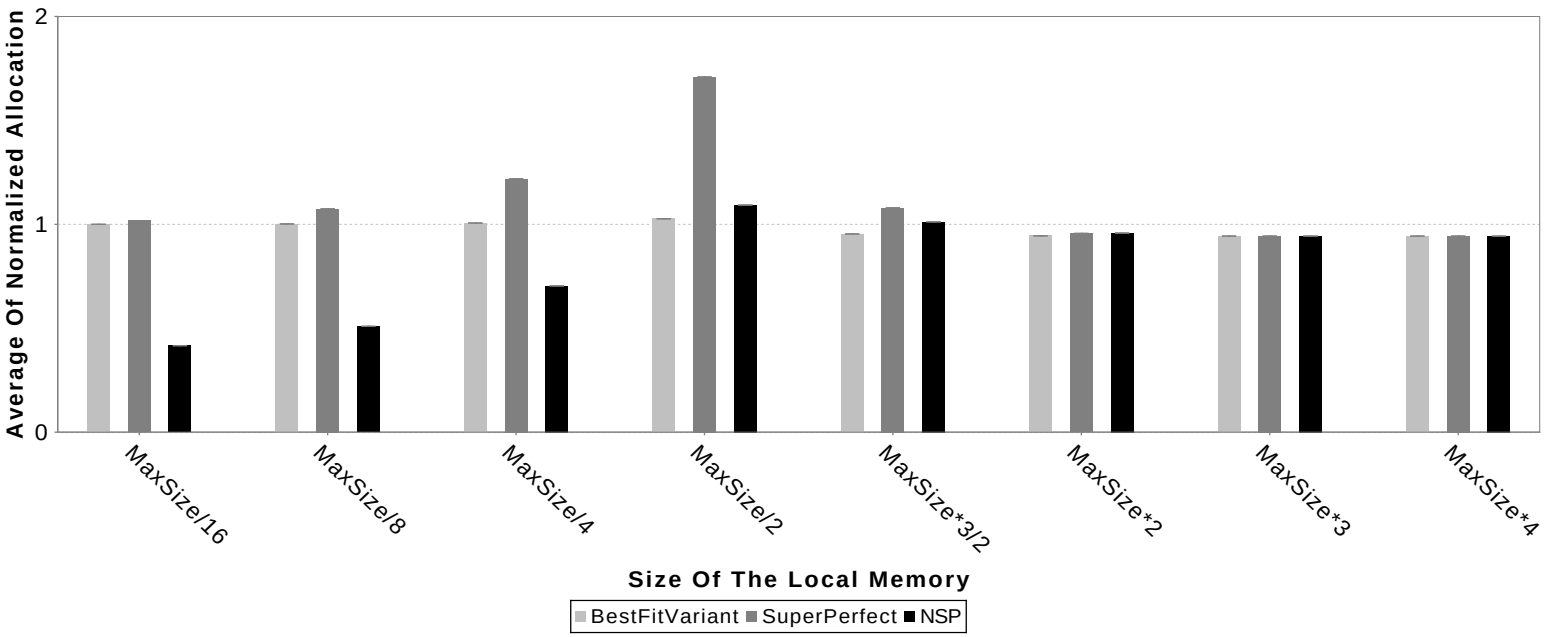
## Graph Generation

- 1000 of superperfect graphs
- 1000 of arbitrary graphs

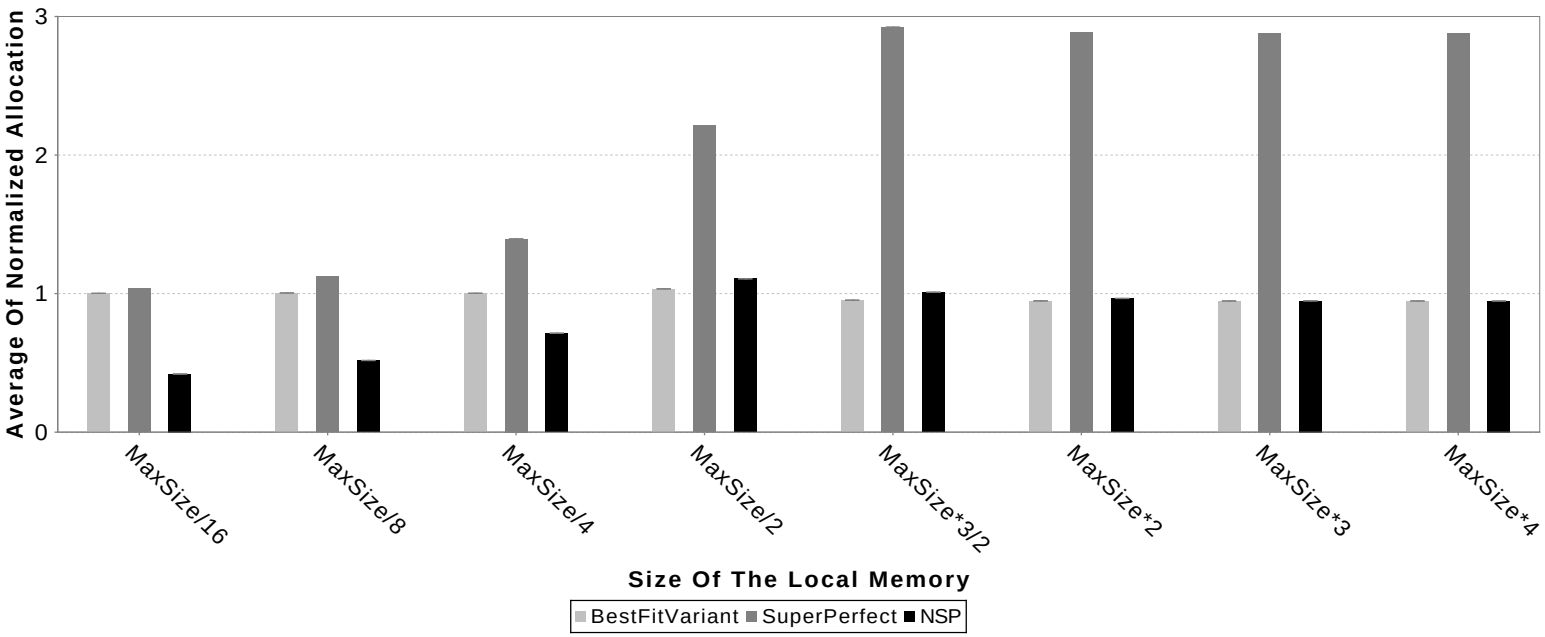
## Compared Algorithms

- `BestFit`
- `BestFitVariant` (LM copy avoidance)
- `SuperPerfect` (state of the art approach [Li'11])

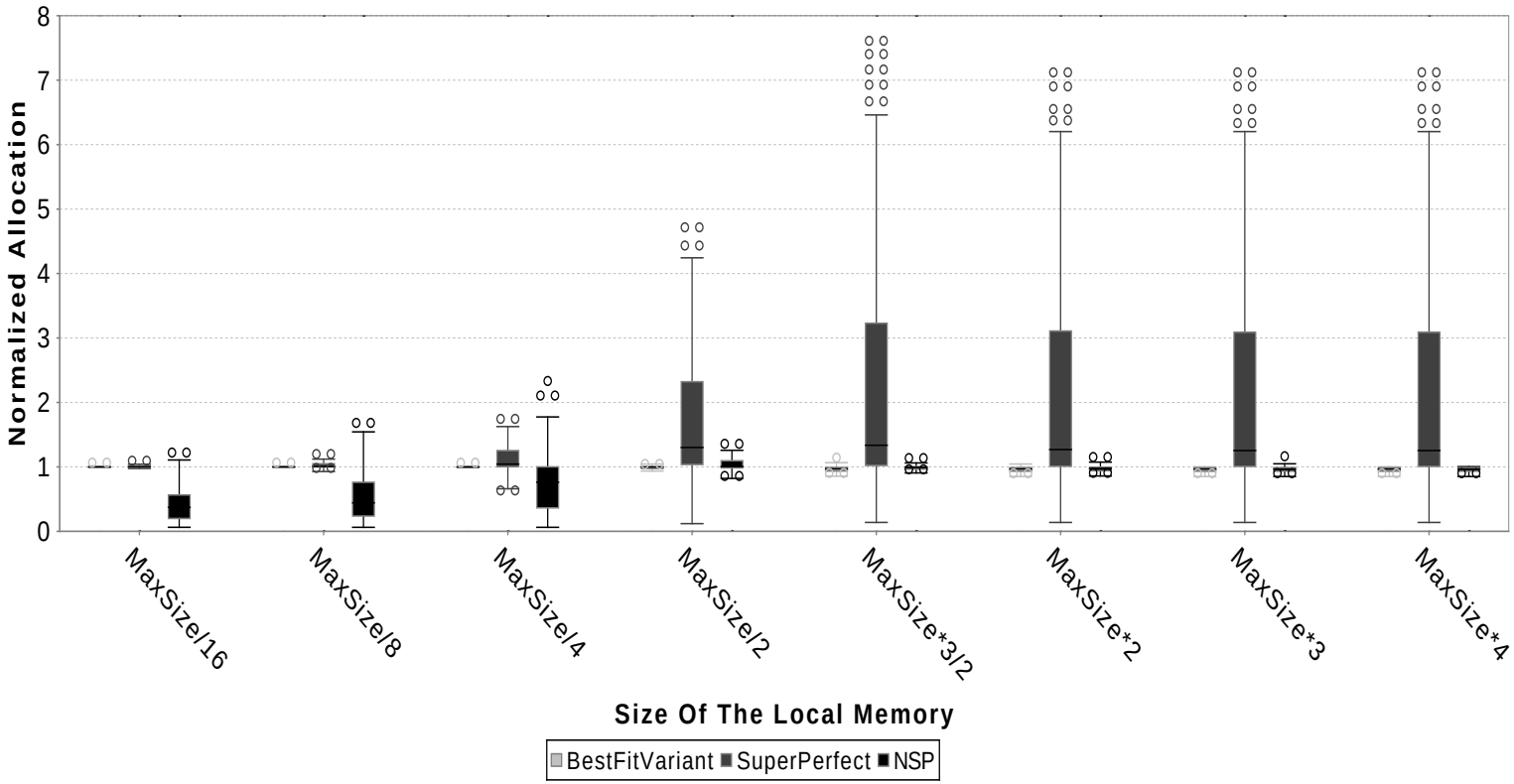
# SuperPerfect Graphs



# Arbitrary Graphs



# Arbitrary Graphs



# Outline

Introduction

Weighted Interval Graph Coloring

LM Allocation through NSP WIG Coloring

Experimental Evaluation

Conclusion

# Conclusion and Perspectives

## Conclusion

- Submarine-building problem: strong and novel complexity results
- Approximation algorithm
- Decoupling of spill code generation and from assignment
- Experimental evaluation shows very favorable results compared to state-of-the art allocators

## Perspectives

- Extend the work to environments where many threads share the same LM
- Consider programming models like (HMPP, OpenCL) offering more support for software-controlled local memories to PGAS (Partitionned Global Address Space) languages requiring more attention to the memory locality