A Transducer-Based XML Query Processor

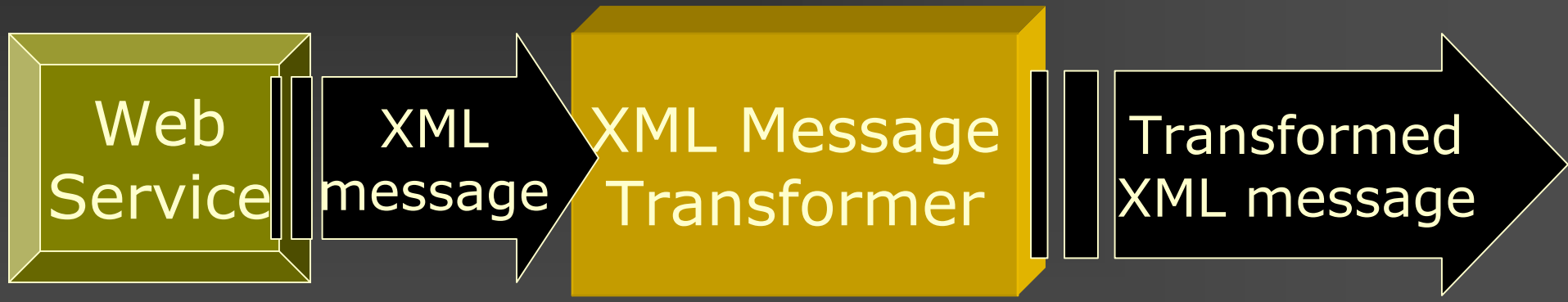
Bertram Ludäscher, SDSC/CSE UCSD
Pratik Mukhopadhyay, CSE UCSD
Yannis Papakonstantinou, CSE UCSD

Overview

- Motivation
- Architecture
- Framework: Streams + XQuery
- XSM (XML Stream Machine)
- XSM Networks
- Network Composition
- Conclusions

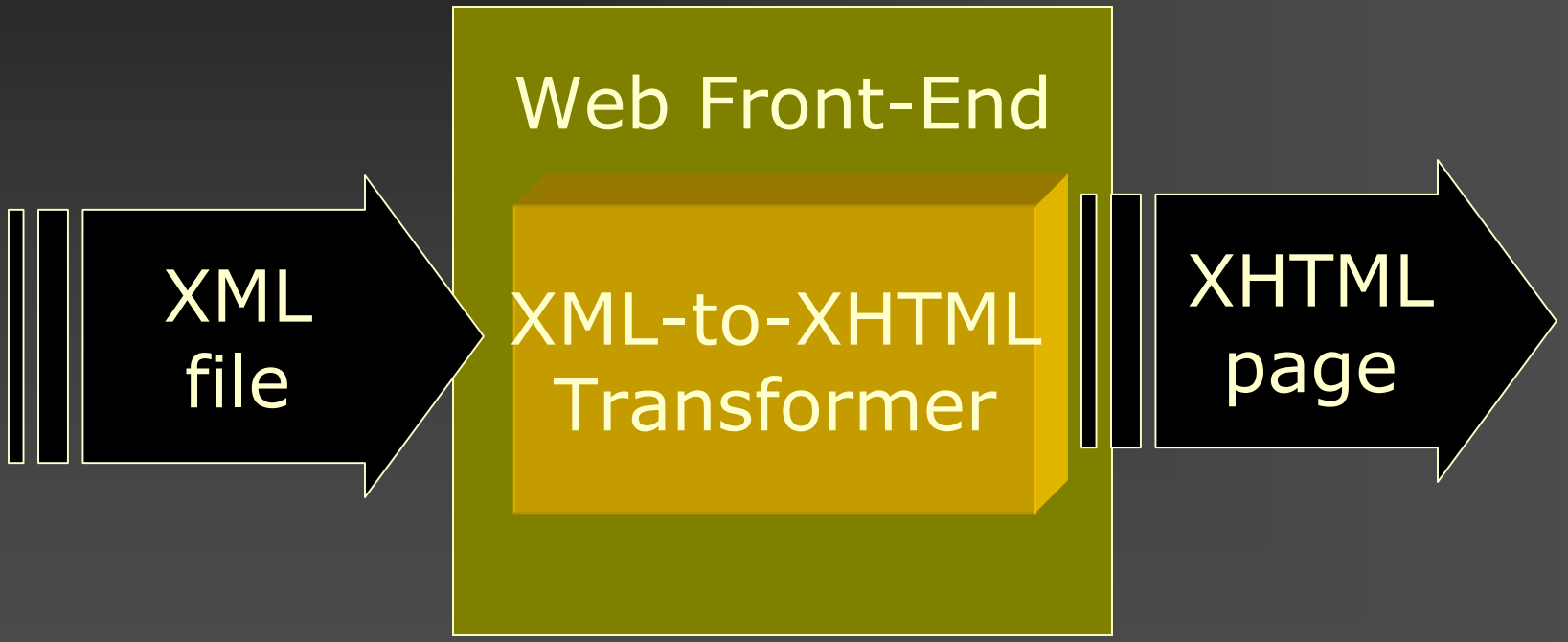
Efficient Processing of Sequentially Accessed XML Data

Web Service Implementations
& RMI



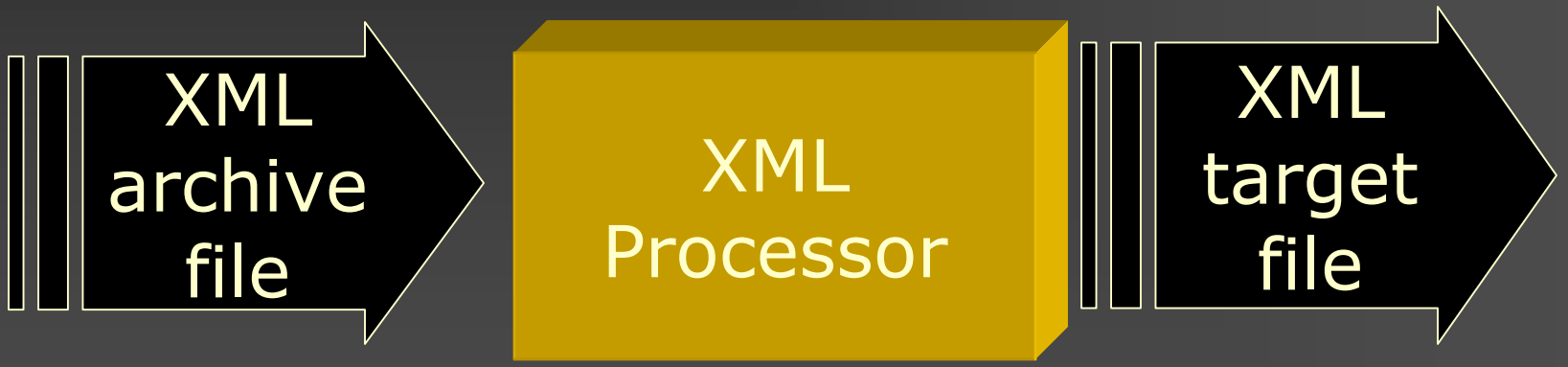
Efficient Processing of Sequentially Accessed XML Data

Web Development



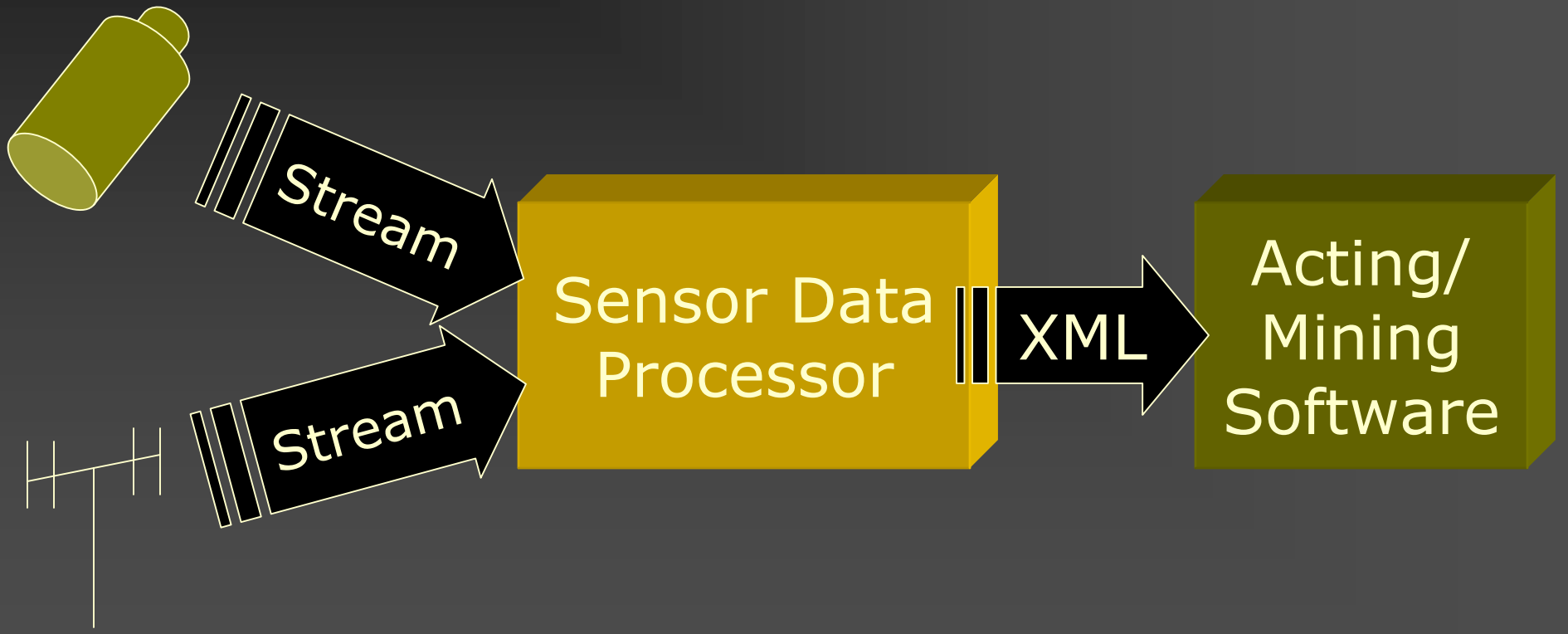
Efficient Processing of Sequentially Accessed XML Data

Archive Transformation &
ETL (Extraction Transformation & Loading)
Applications

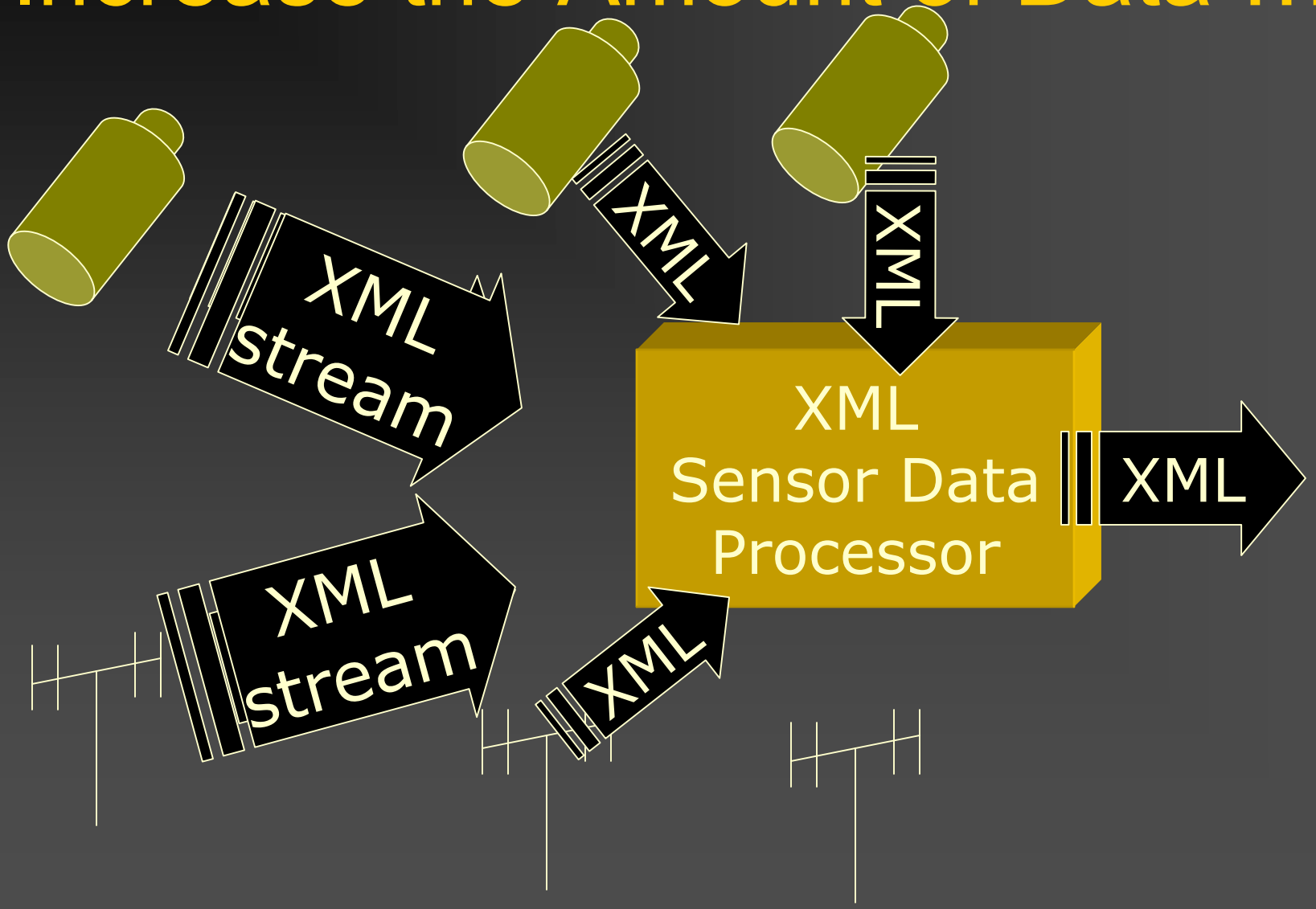


Efficient Processing of Sequentially Accessed XML Data

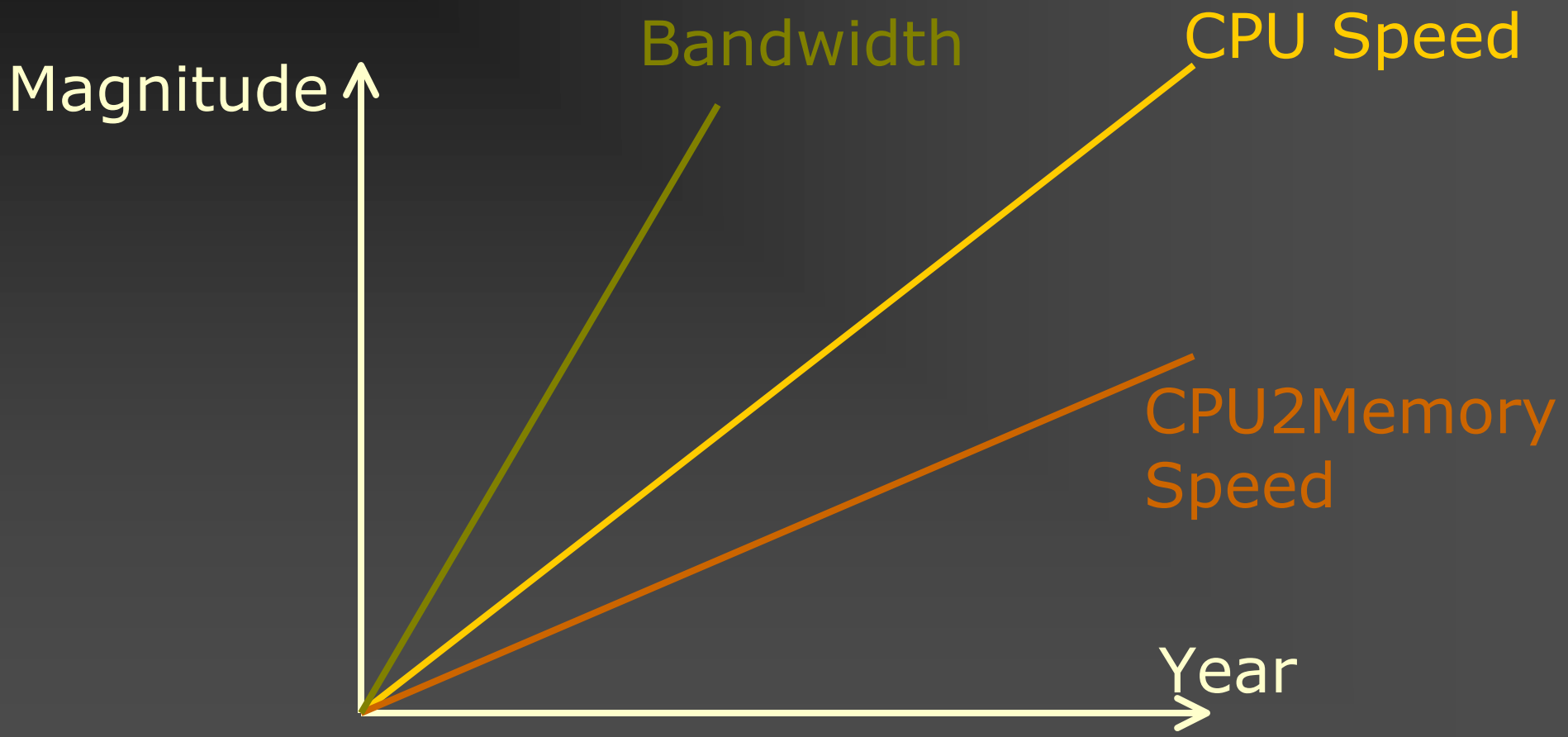
Sensor Data Analysis



Bandwidth & Connectivity will Increase the Amount of Data …



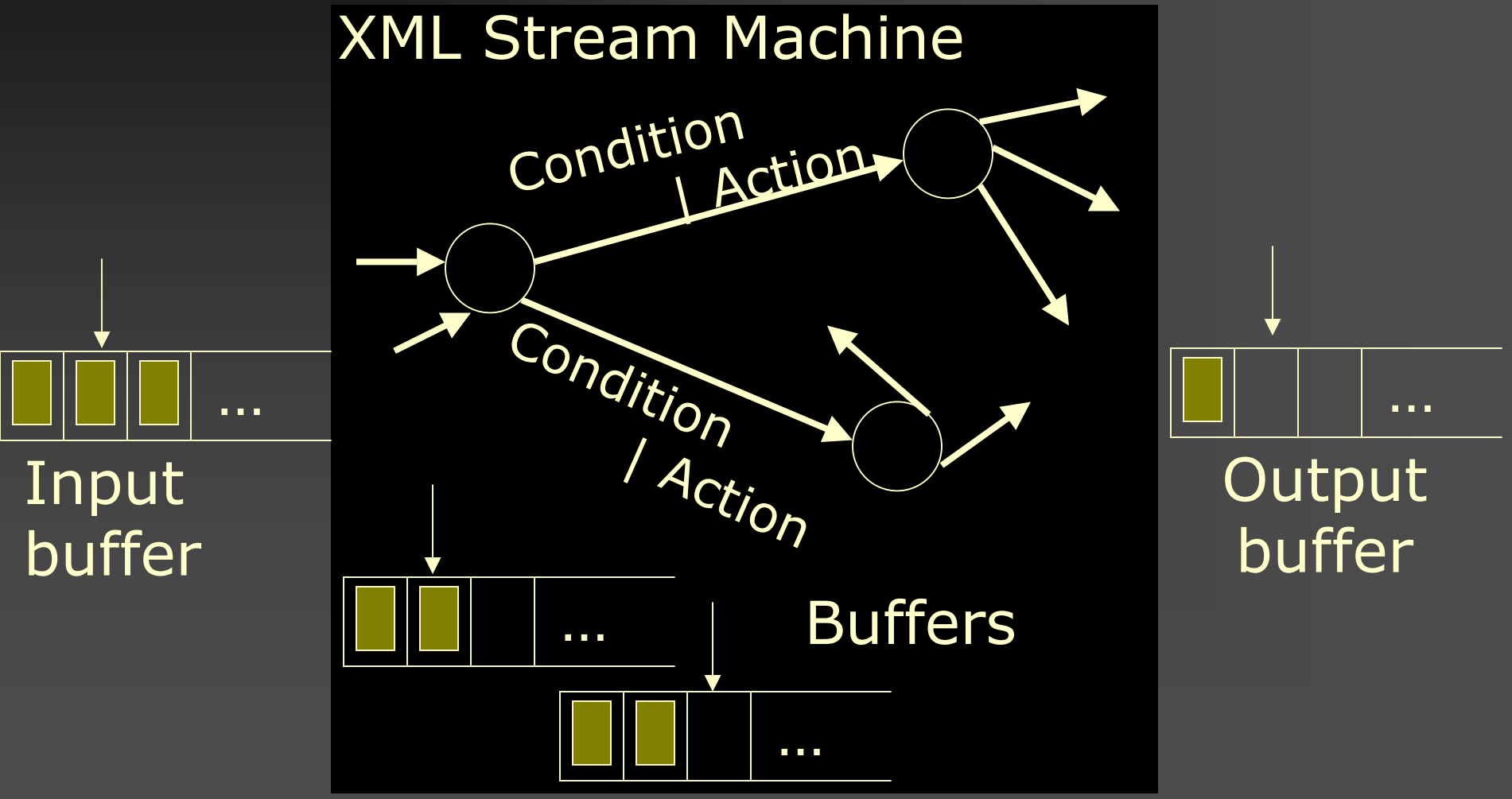
…Hardware Advances do not Favor Conventional Architectures



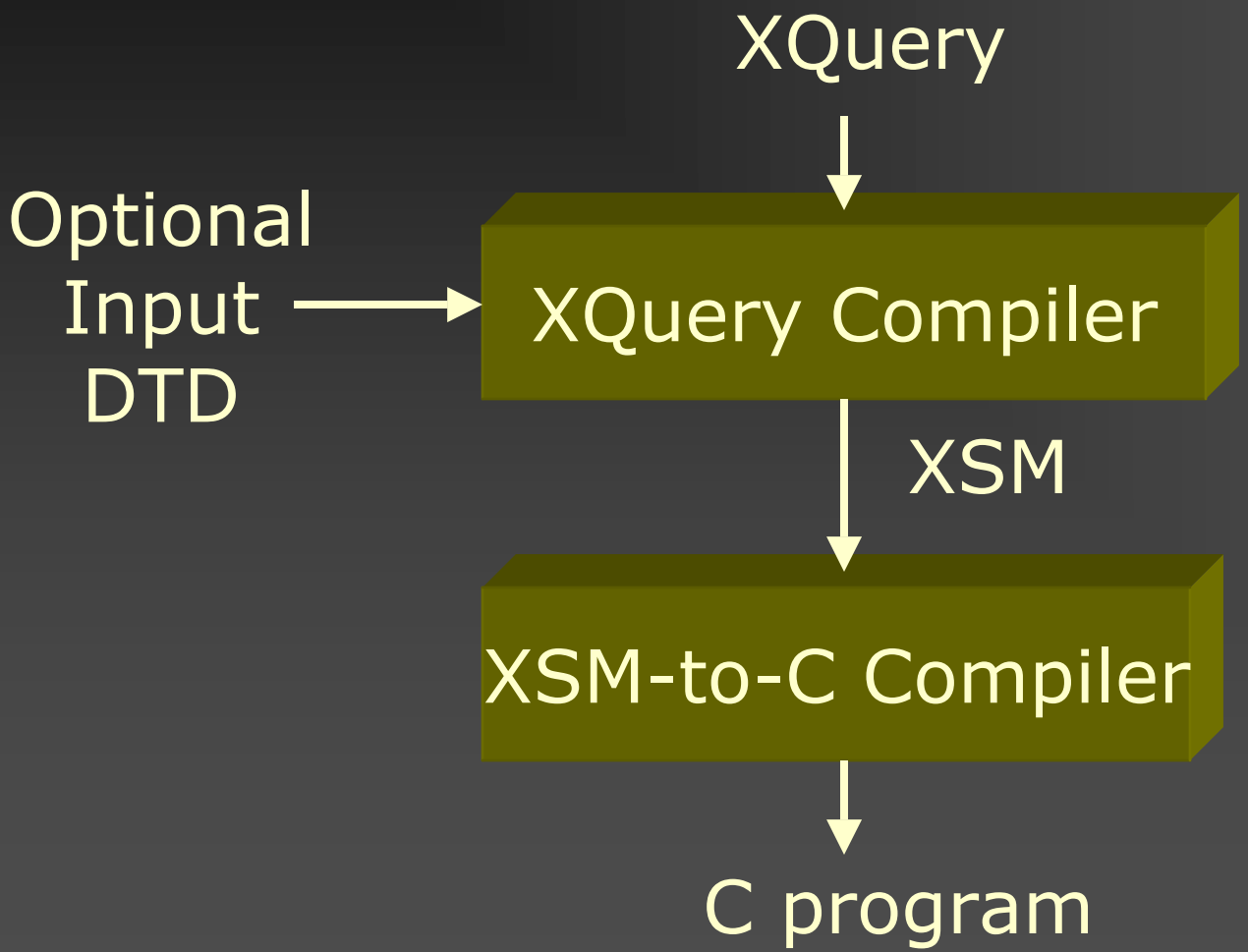
Overview

- Motivation
- Architecture
- Framework: Streams + XQuery
- XSM (XML Stream Machine)
- XSM Networks
- Network Composition
- Conclusions

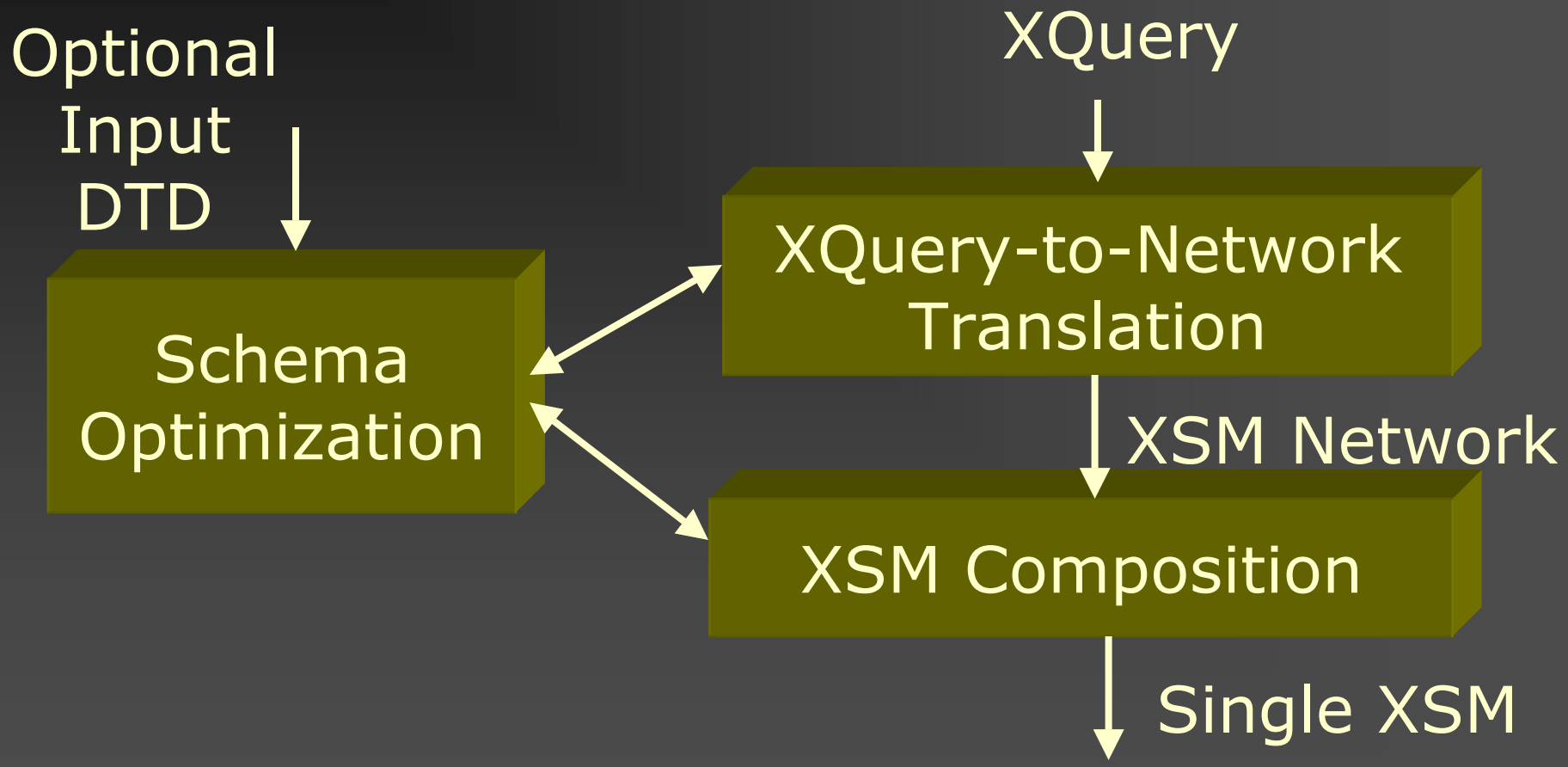
Transducer-Based Processing: On-the-Fly & Minimal Memory



XML Stream Machine (XSM) High-Level Architecture



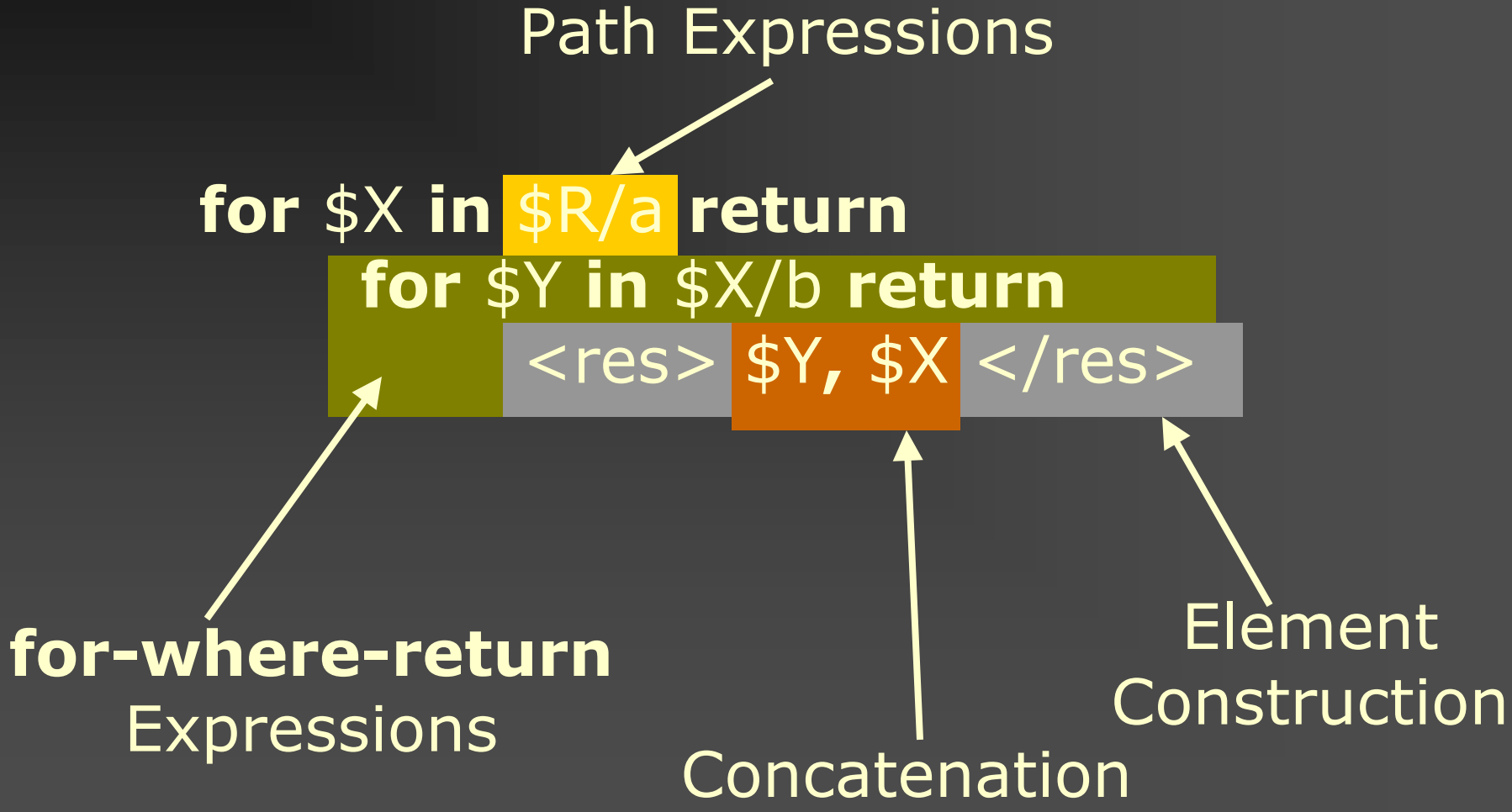
Components of the XQuery Compiler



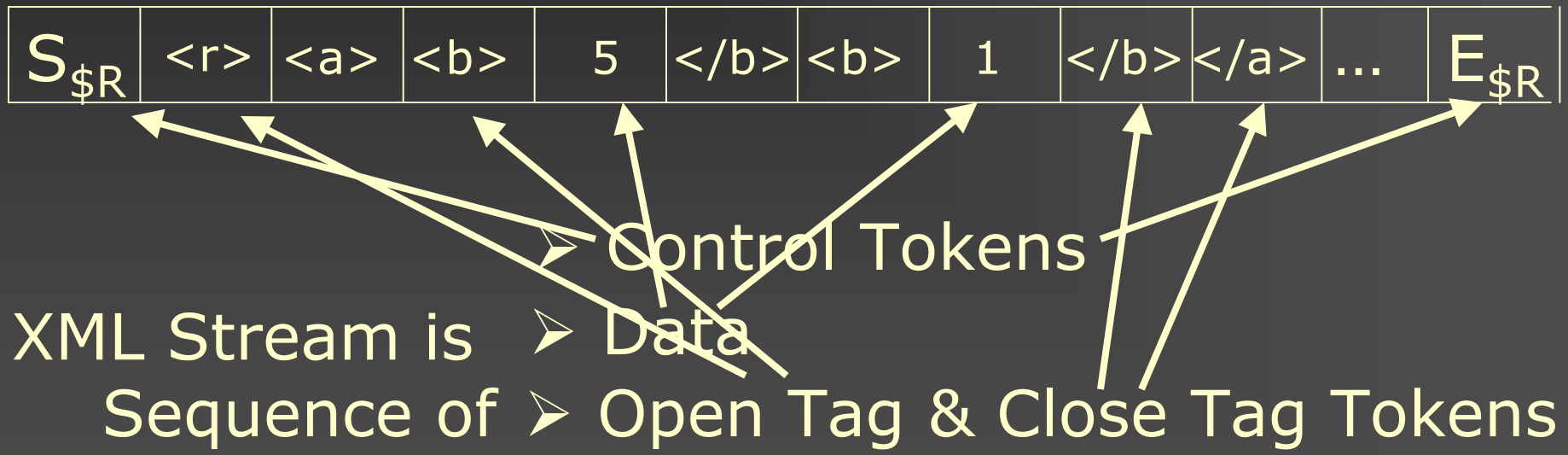
Overview

- Motivation
- Architecture
- Framework: Streams + XQuery
- XSM (XML Stream Machine)
- XSM Networks
- Network Composition
- Conclusions

XQuery Subset



XML Stream: Tags, Data & Control Tokens

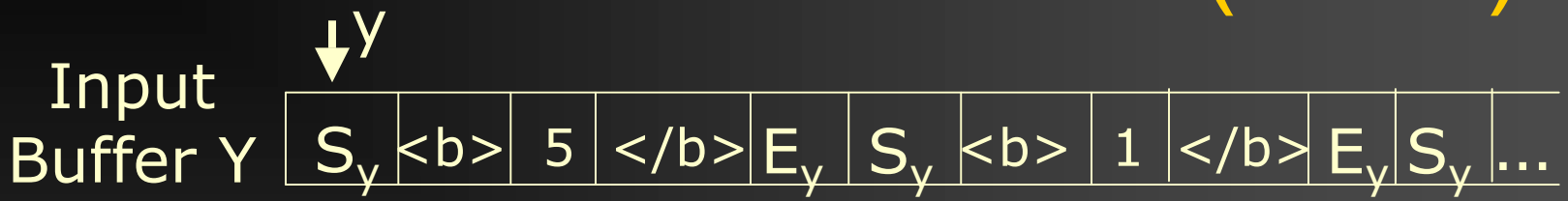


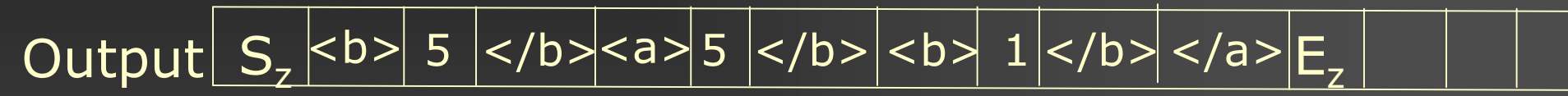
XML Stream is
- Sequence of

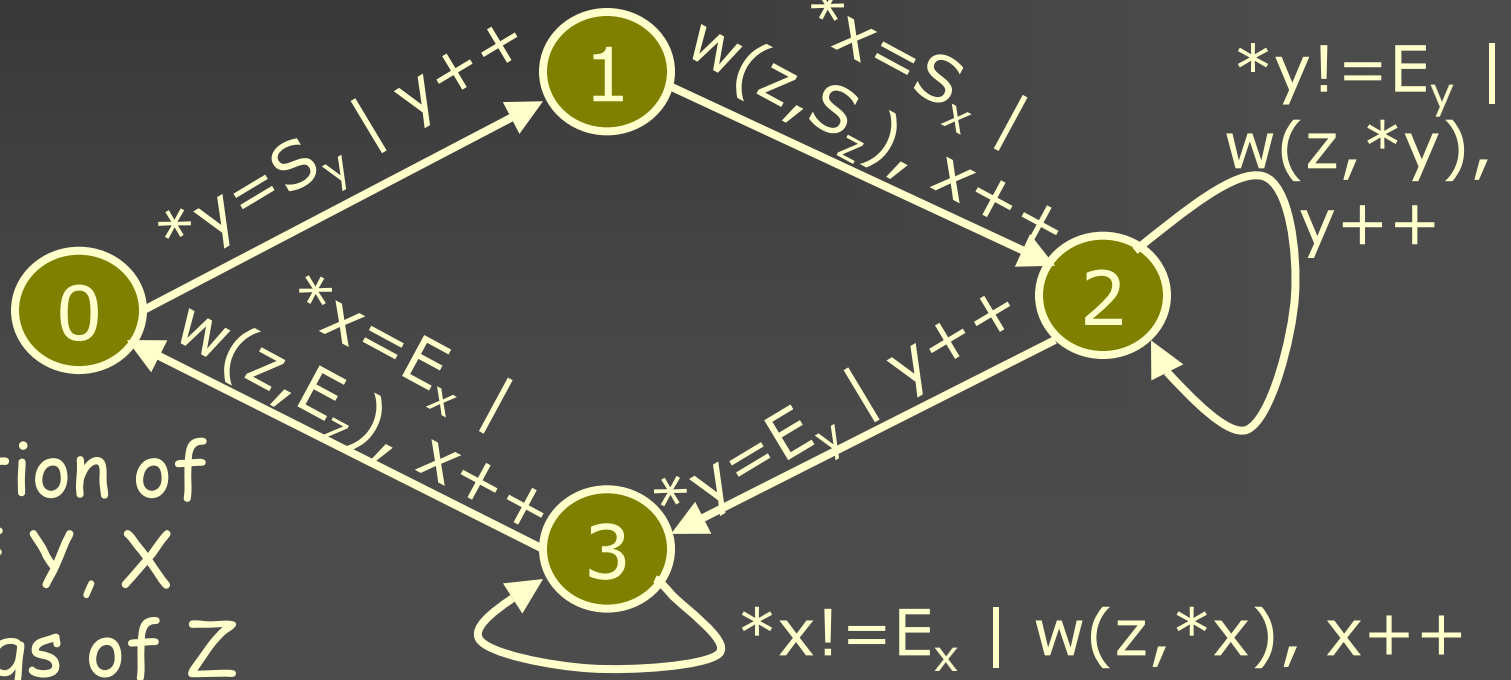
Overview

- Motivation
- Architecture
- Framework: Streams + XQuery
- XSM (XML Stream Machine)
- XSM Networks
- Network Composition
- Conclusions

XML Stream Machine (XSM)



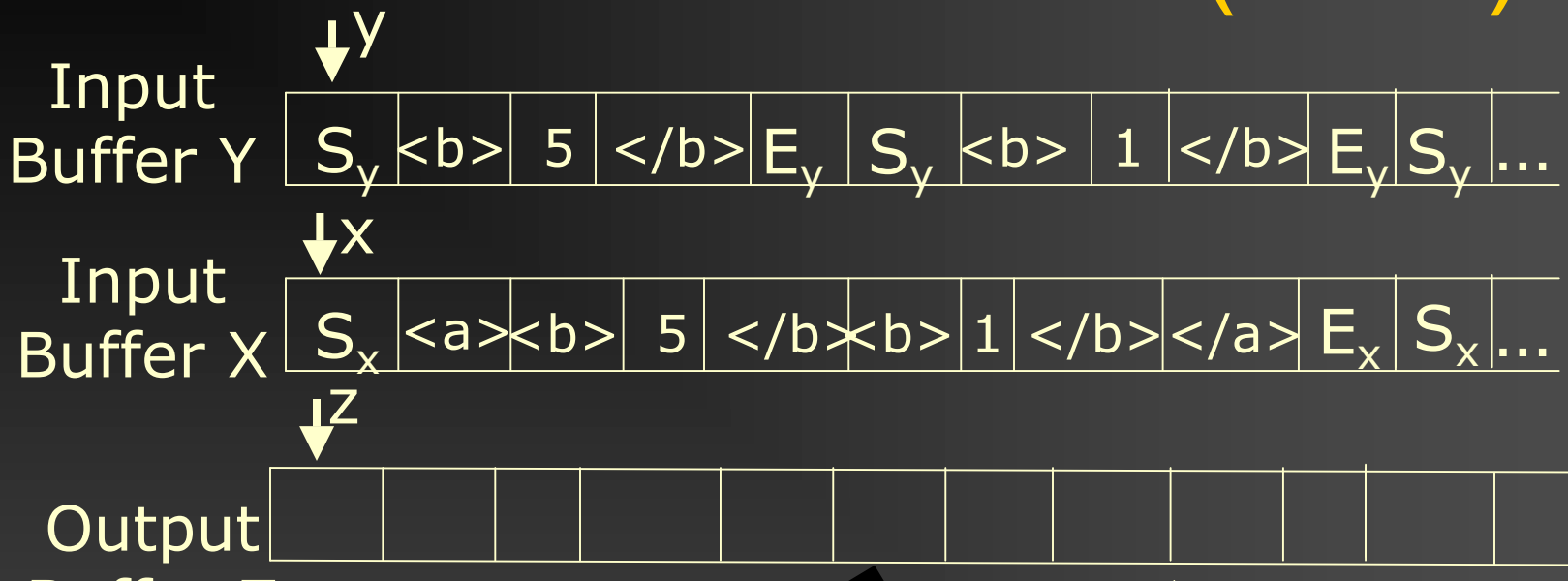


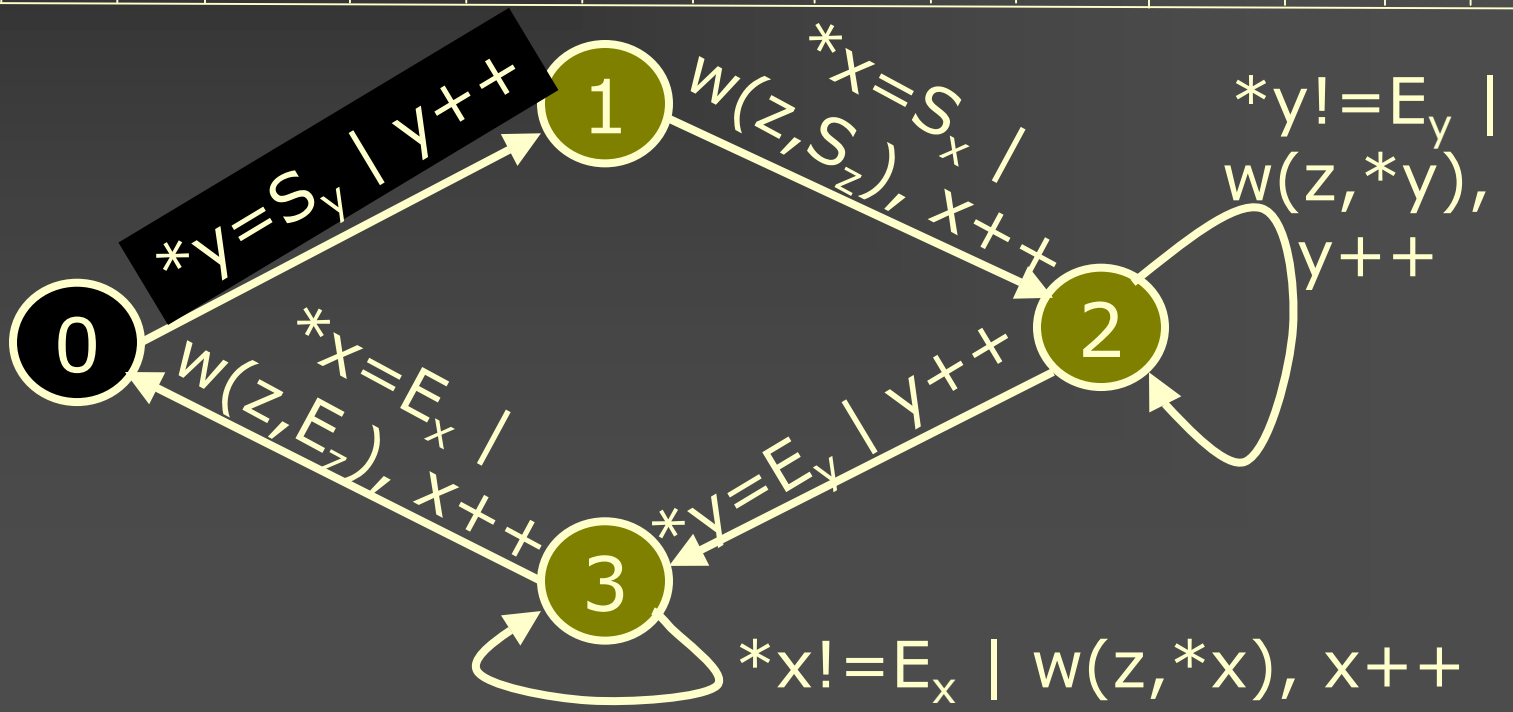


Concatenation of bindings of Y, X into bindings of Z

C

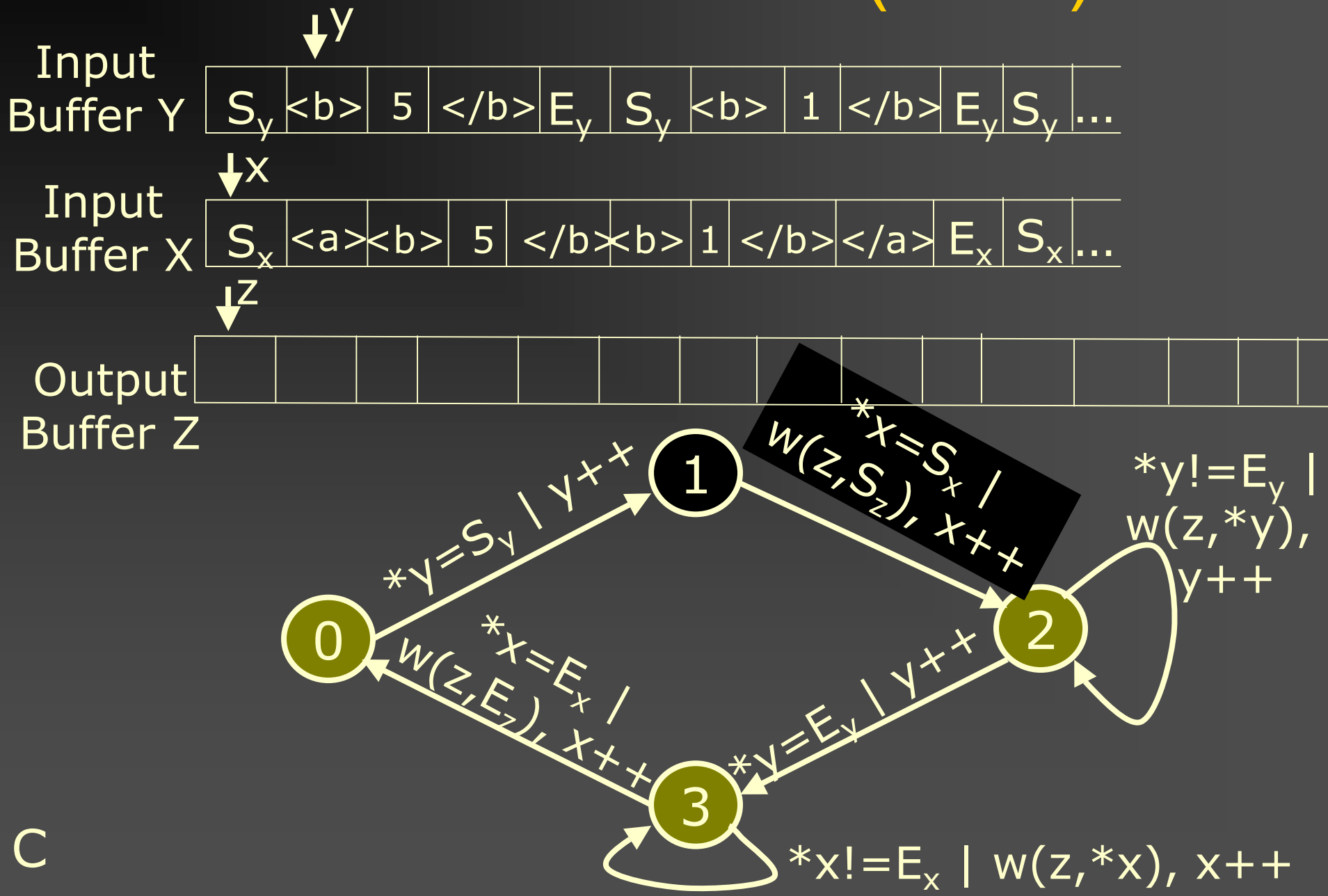
XML Stream Machine (XSM)





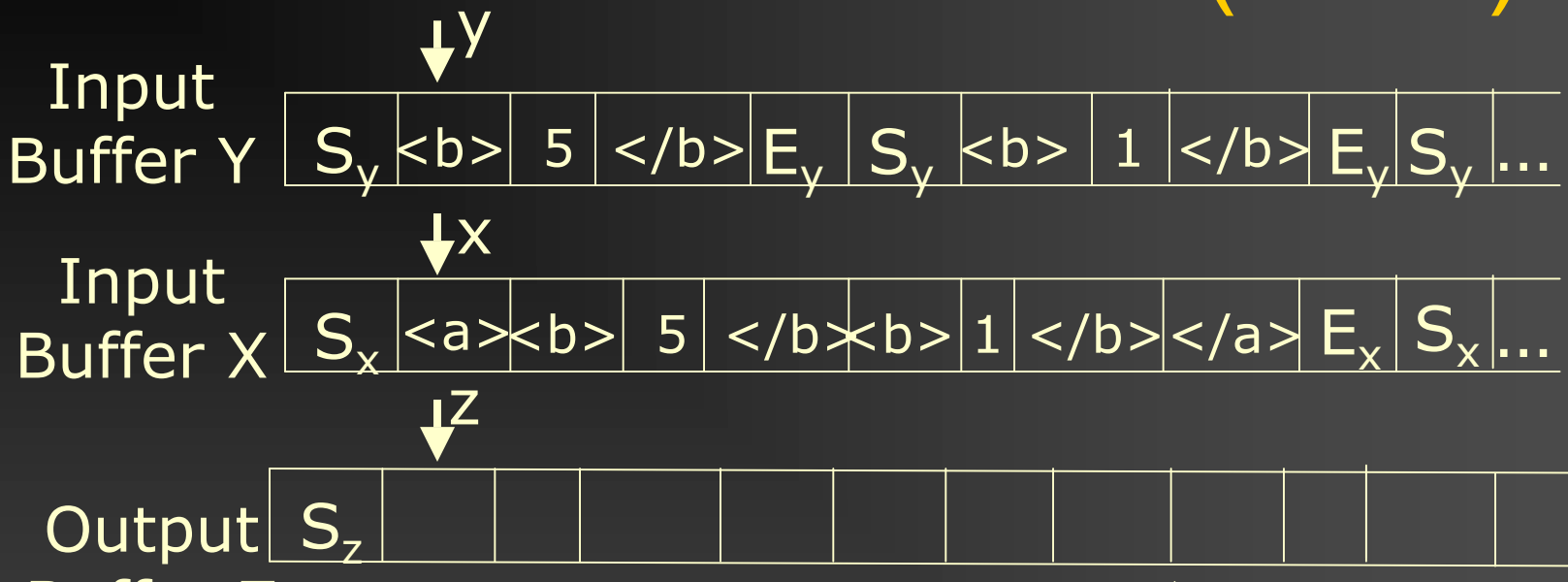
C

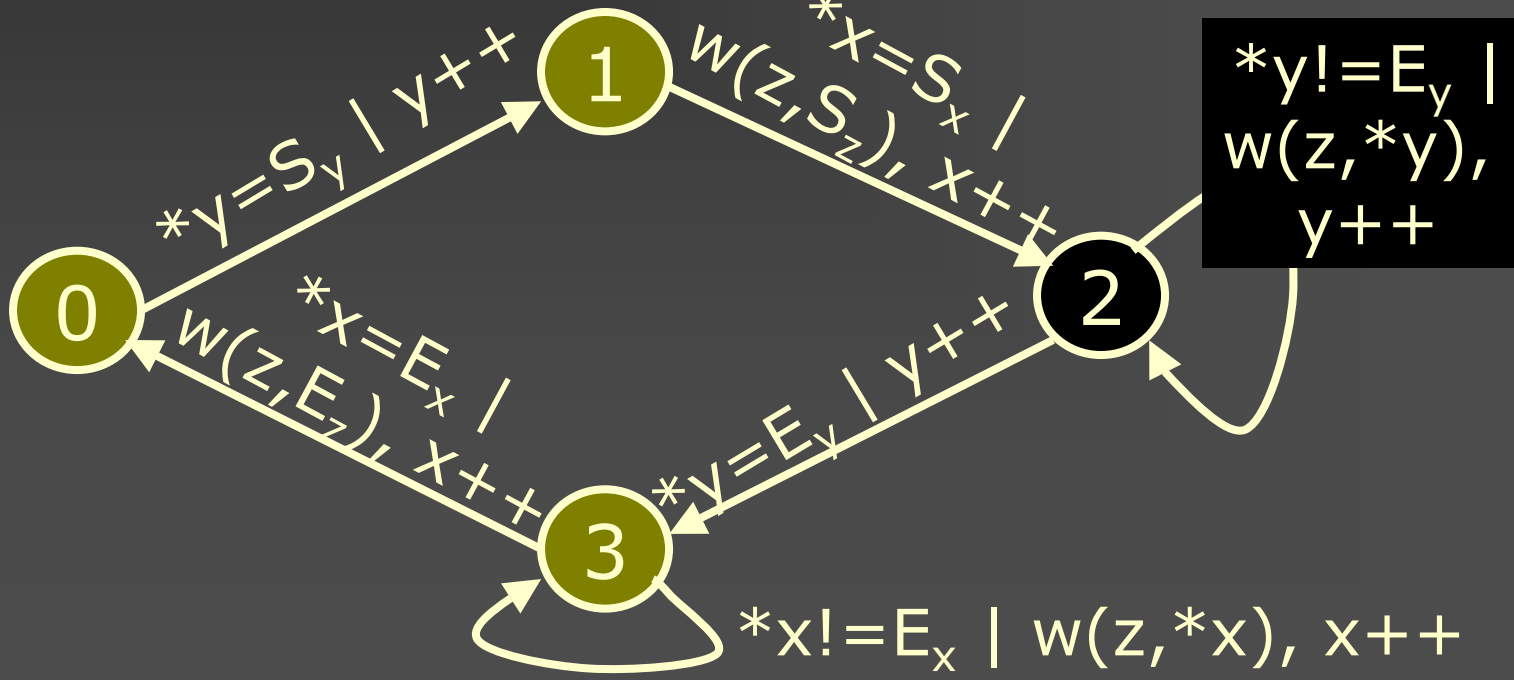
XML Stream Machine (XSM)



C

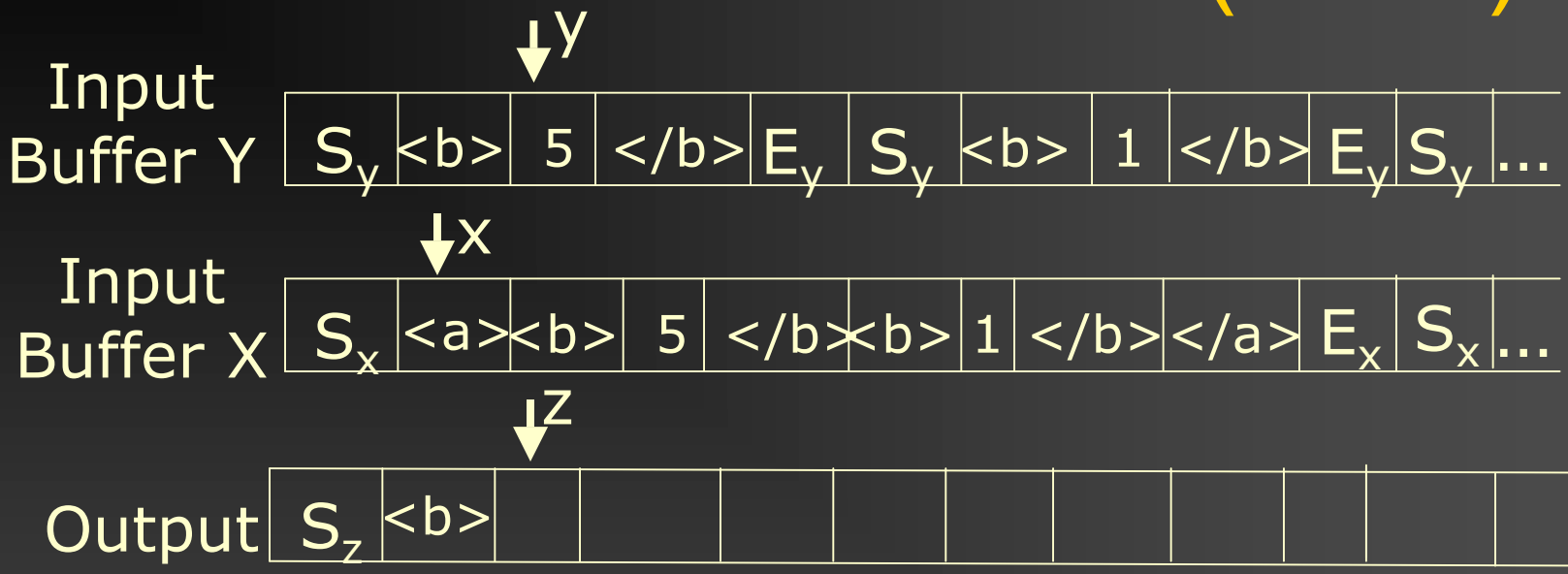
XML Stream Machine (XSM)

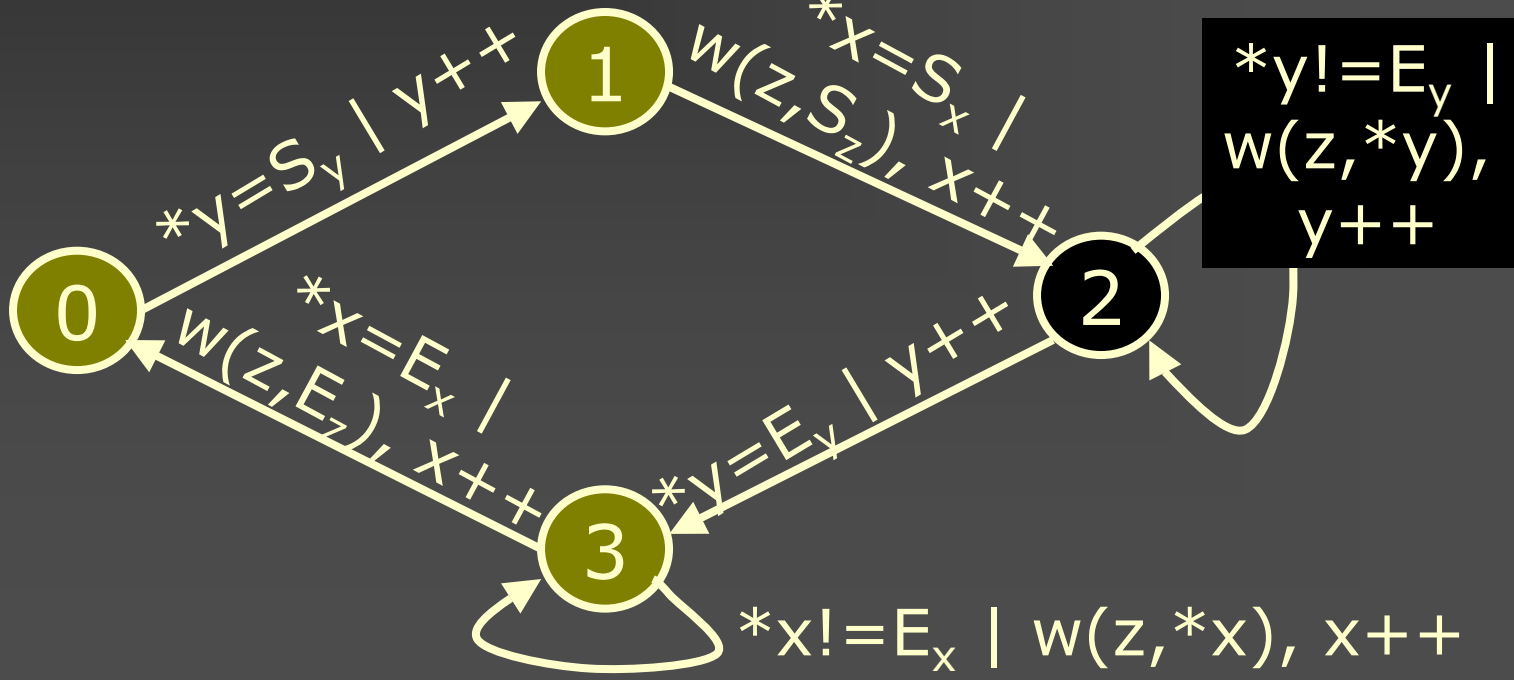




C

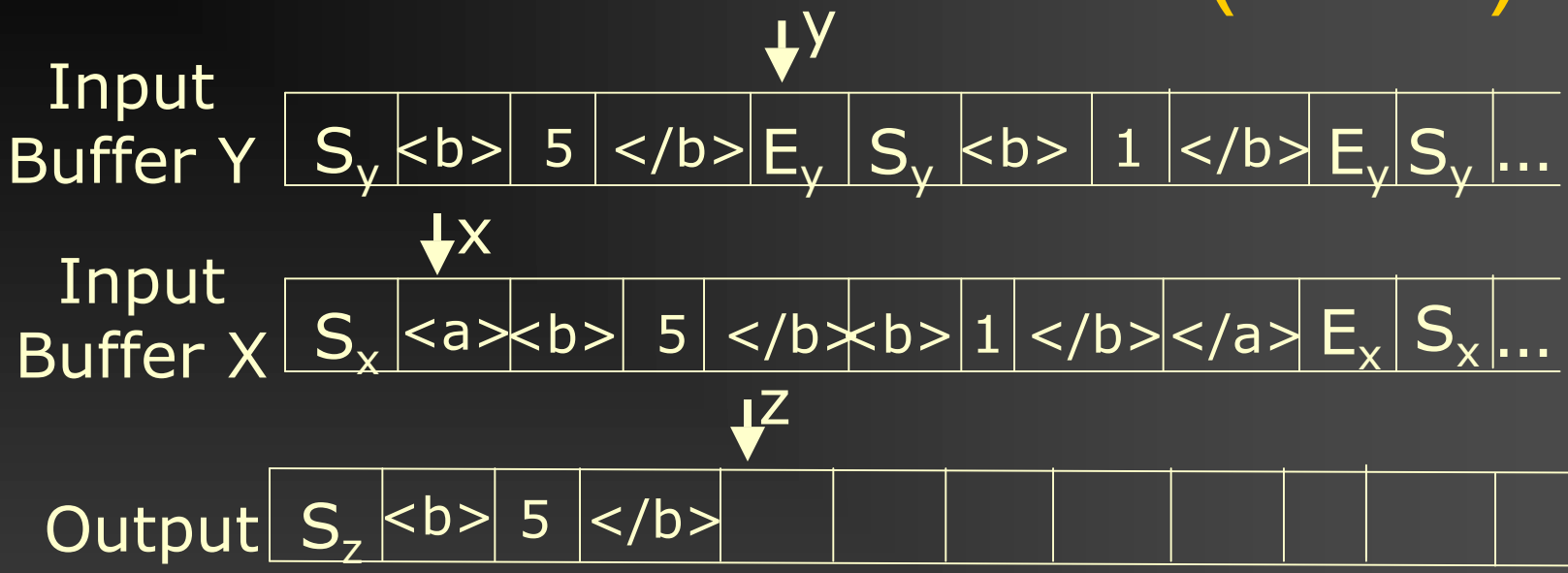
XML Stream Machine (XSM)

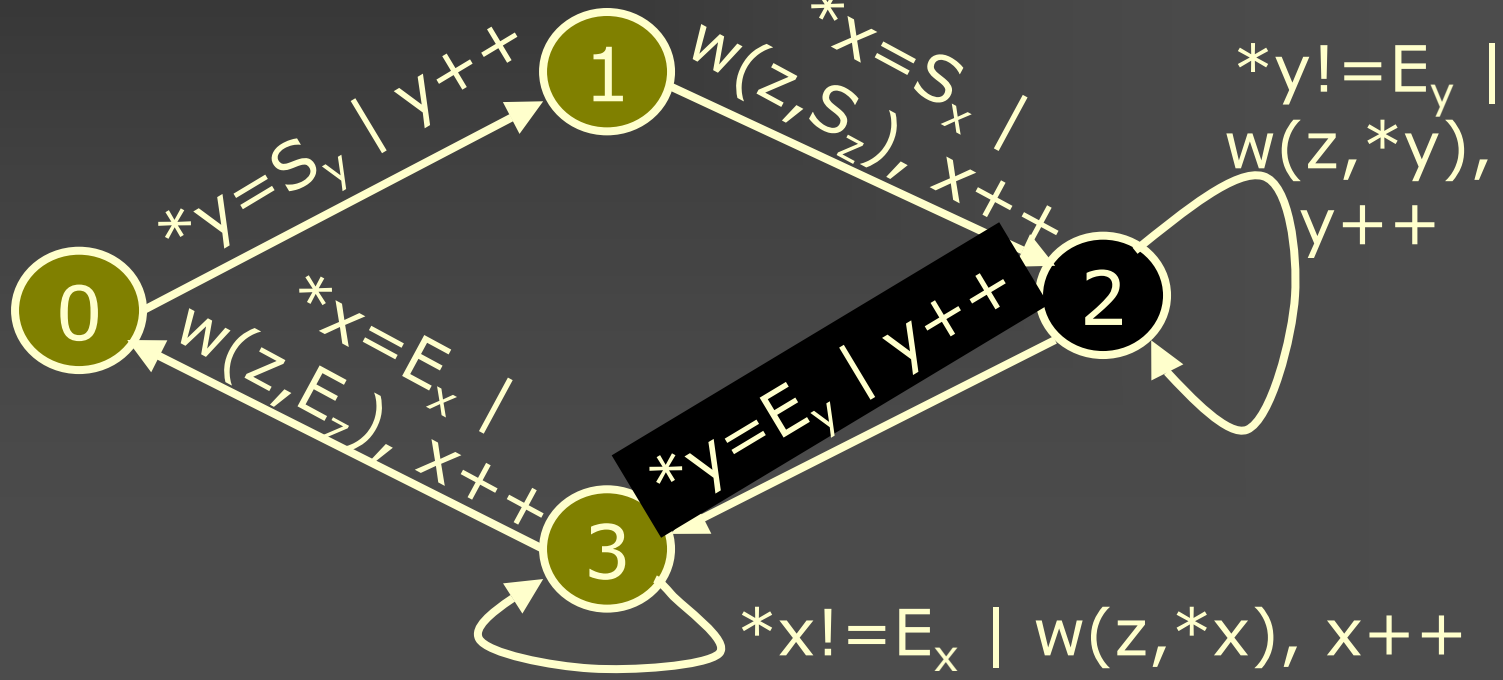




C

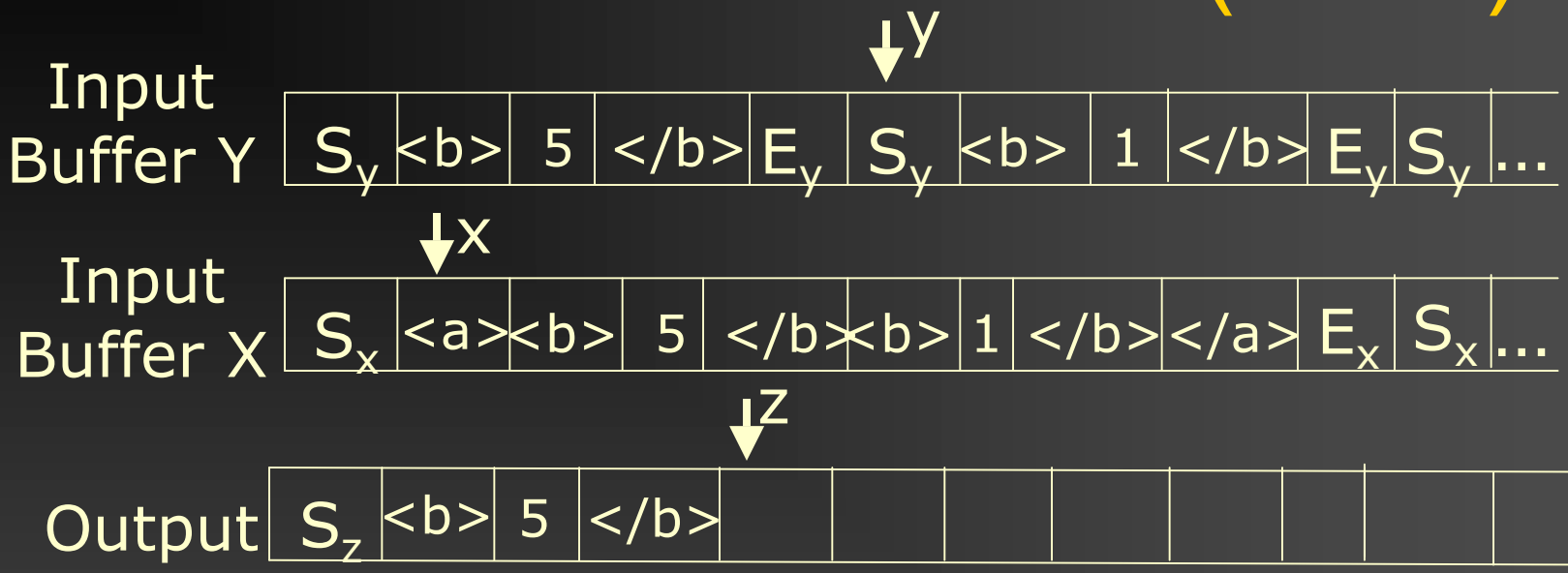
XML Stream Machine (XSM)

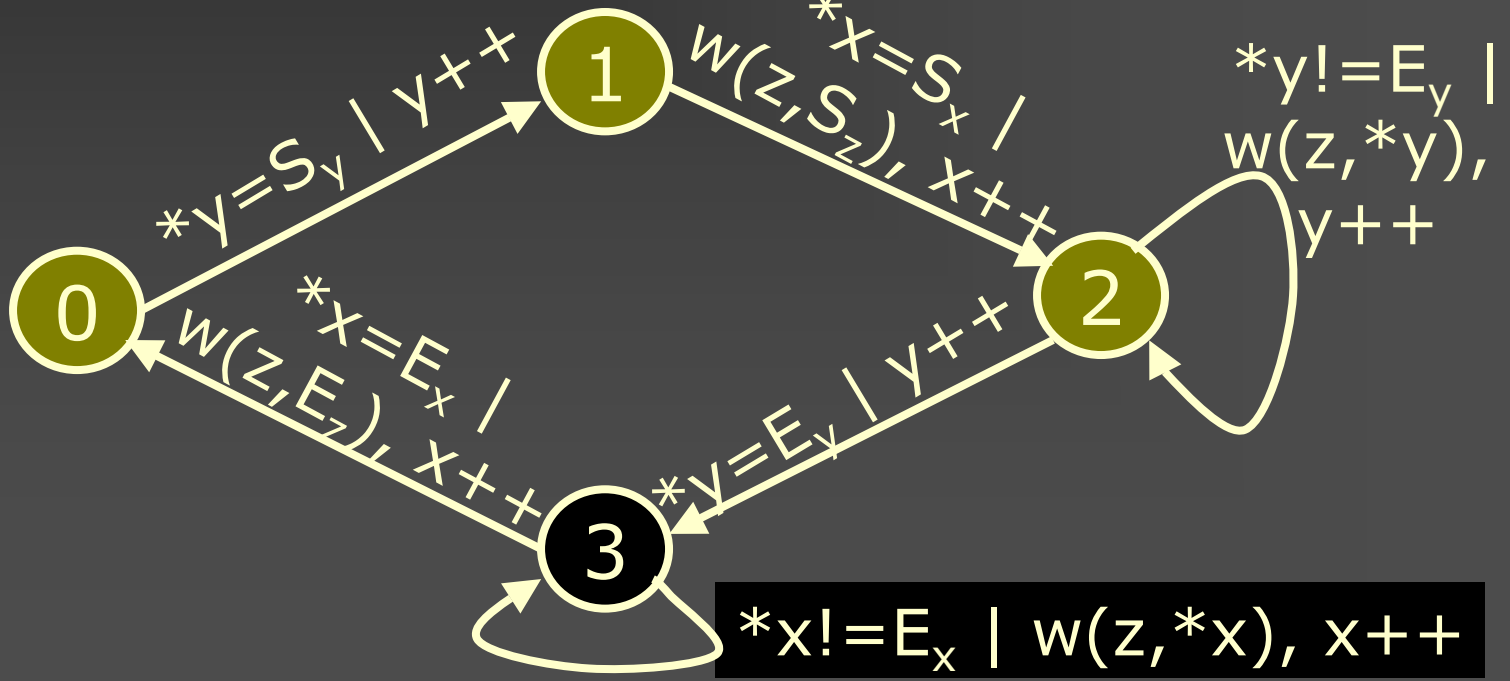




C

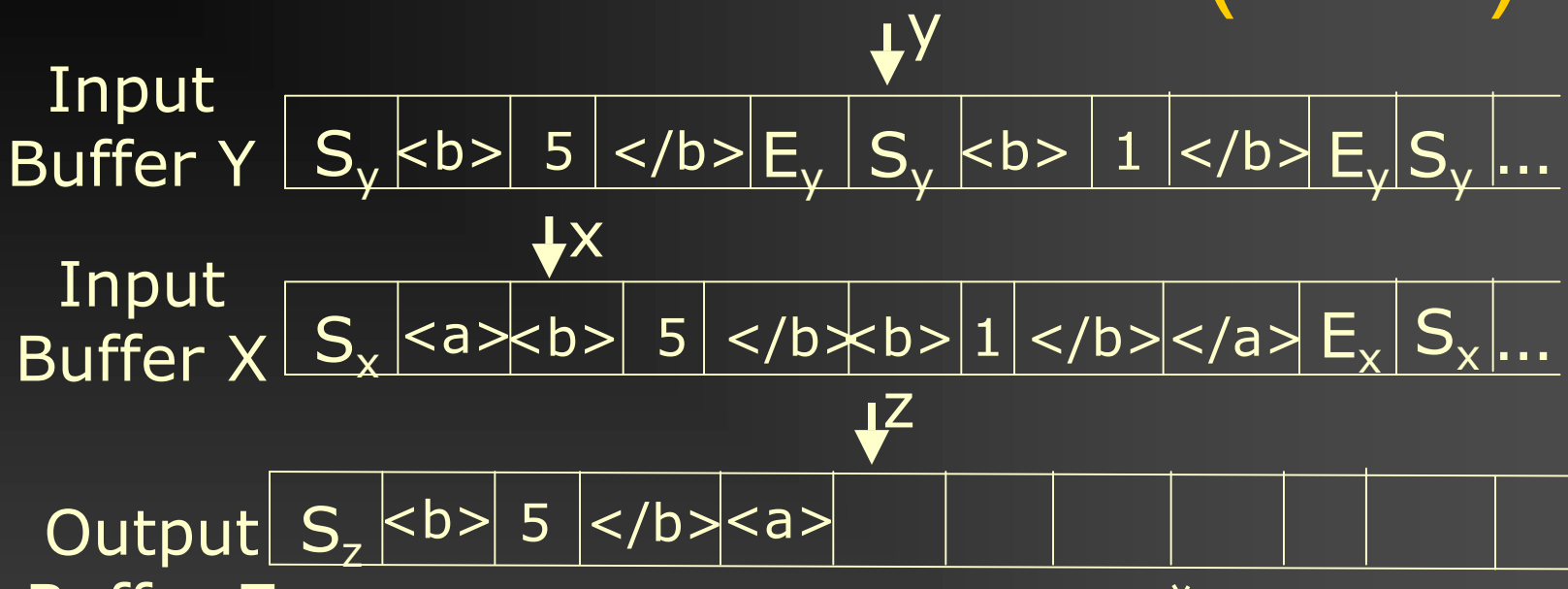
XML Stream Machine (XSM)

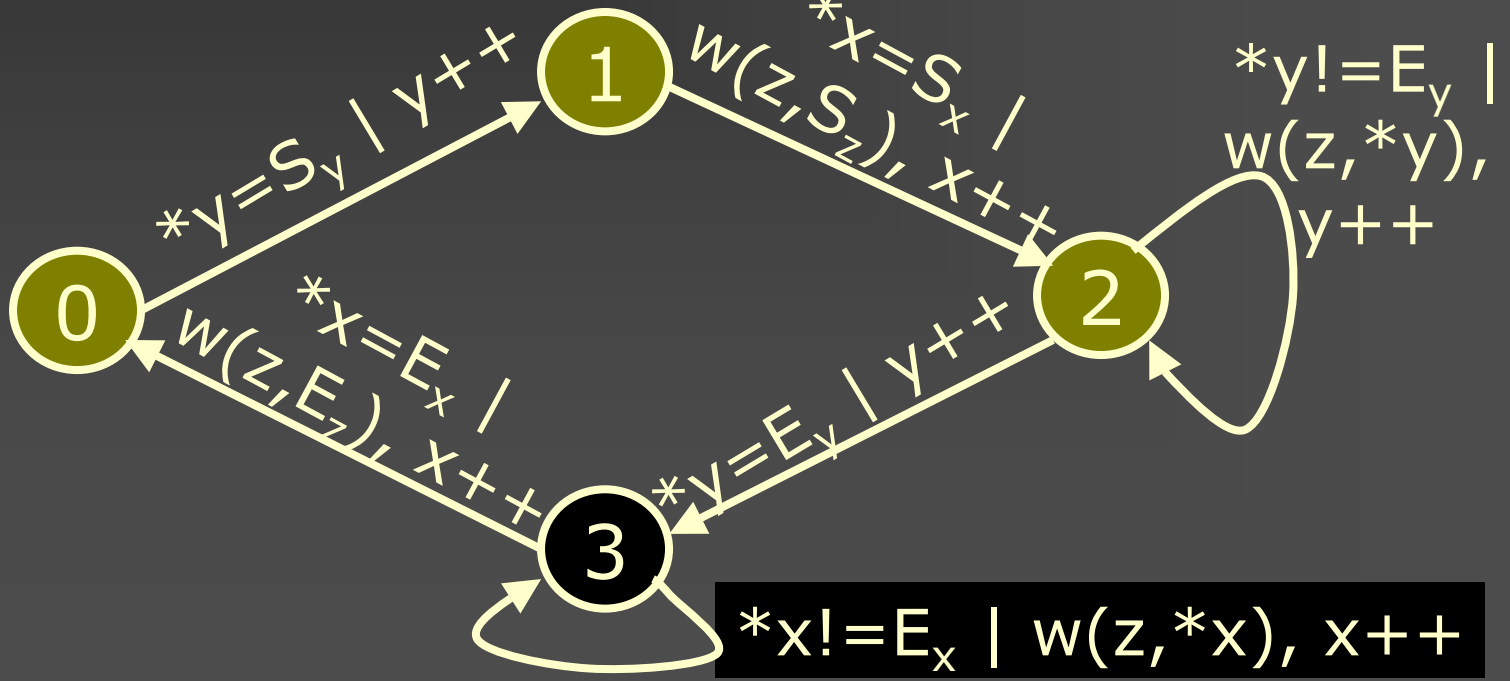




C

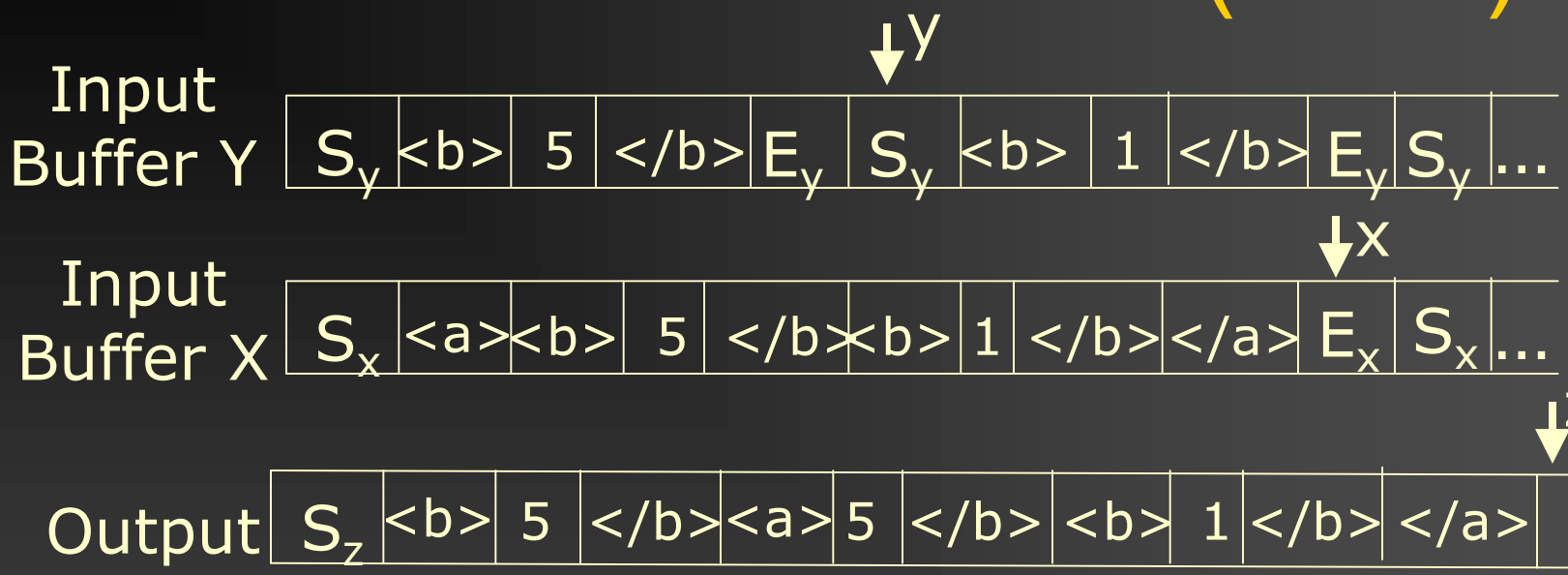
XML Stream Machine (XSM)

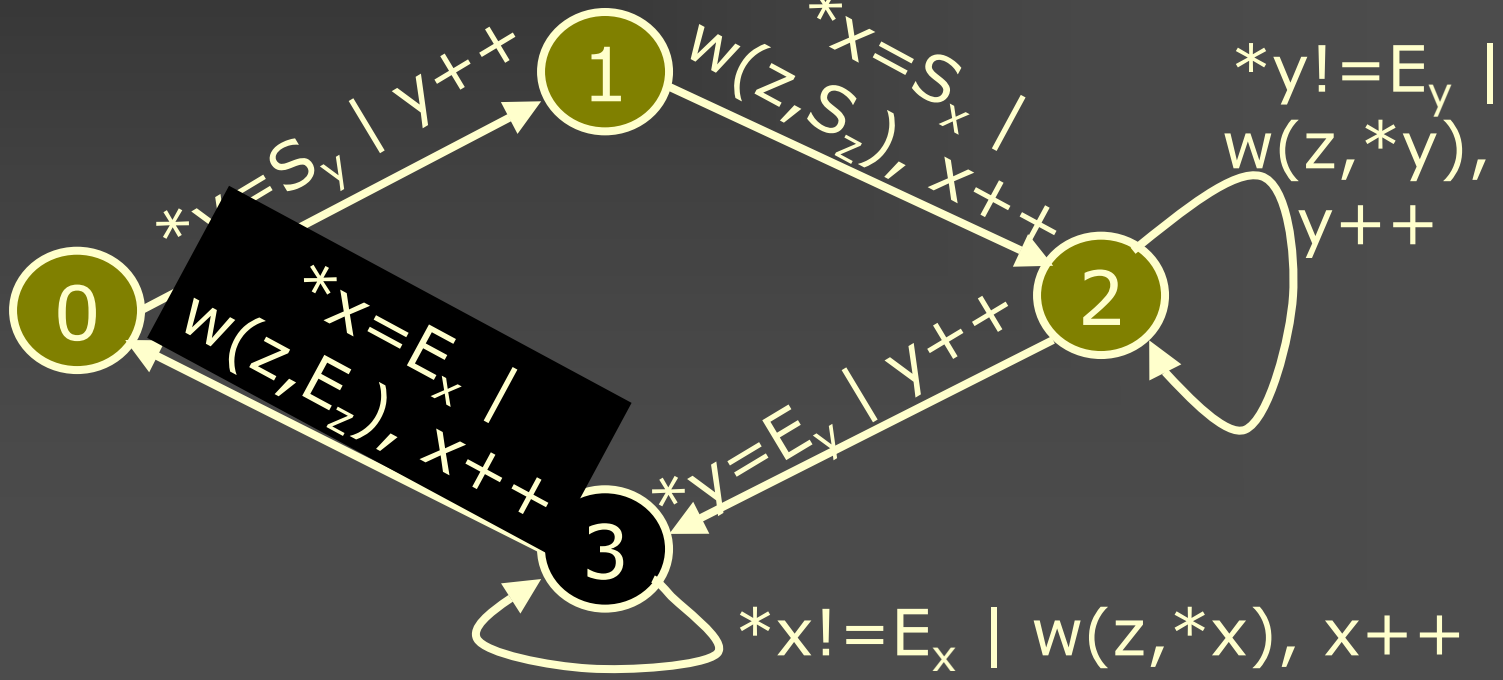




C

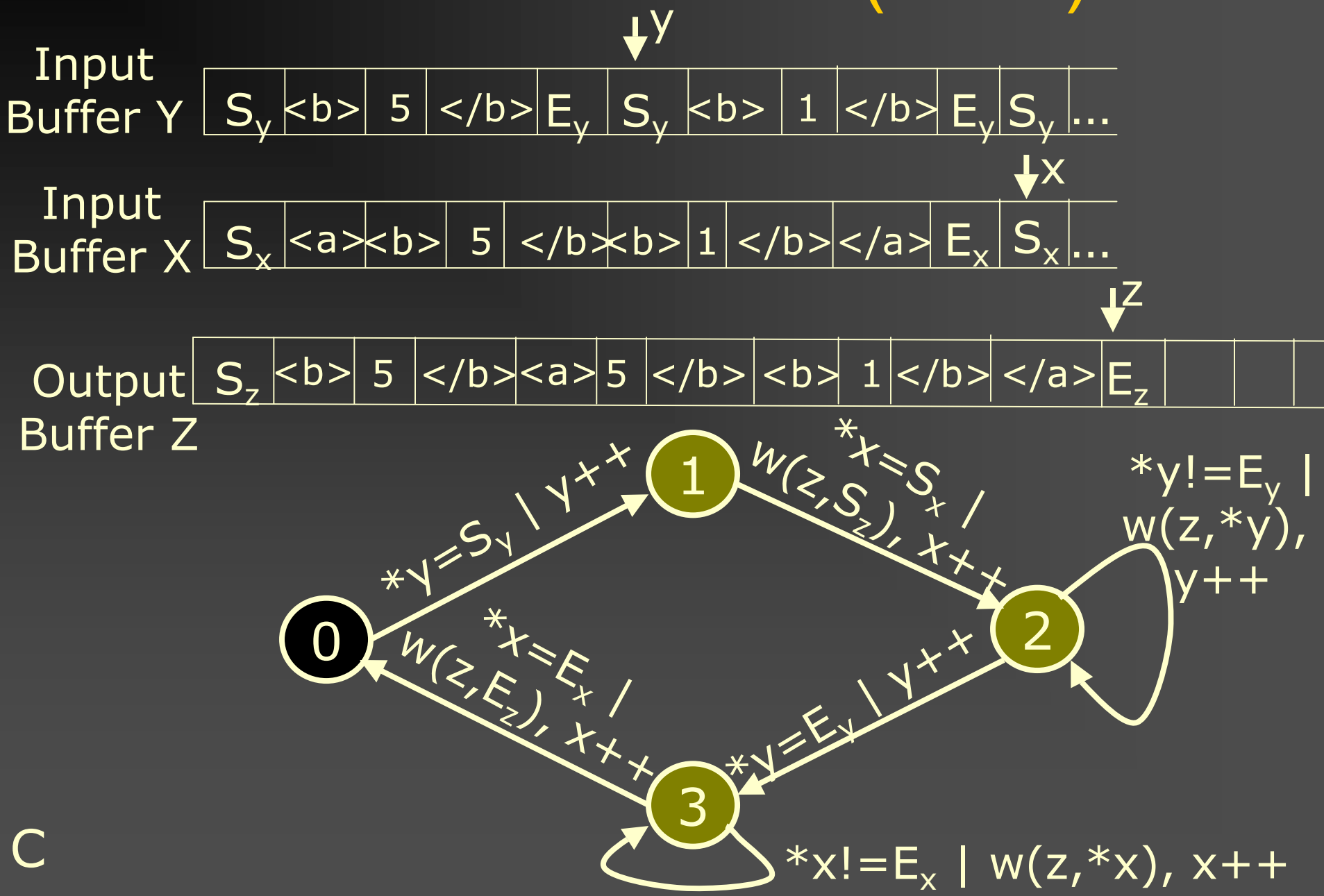
XML Stream Machine (XSM)





C

XML Stream Machine (XSM)



C

Comparison of XSM against State Automata & Transducers

State Automata

- Do not construct
- Do not store intermediate results
- Sufficient for XPath only

Transducers

- Finite alphabets
- State is the only memory
- No reset of input pointers

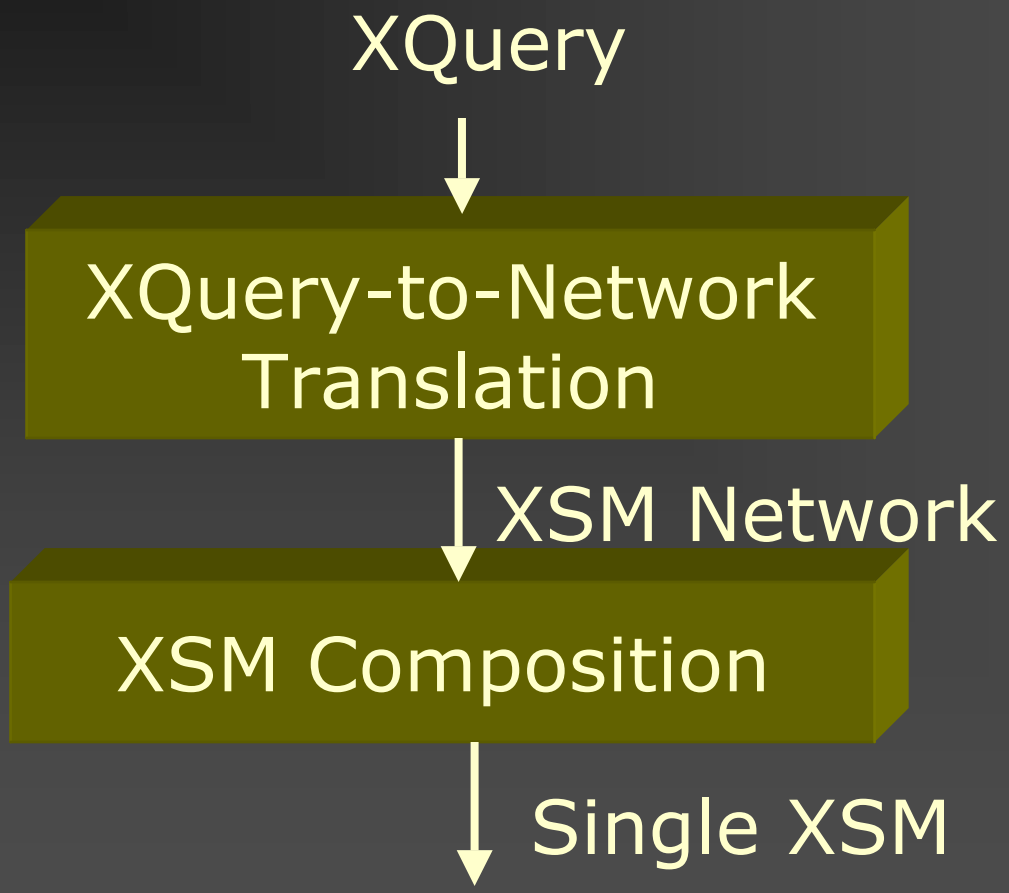
XSM

- Unbounded alphabet
- Buffers
- Pointer reset

Overview

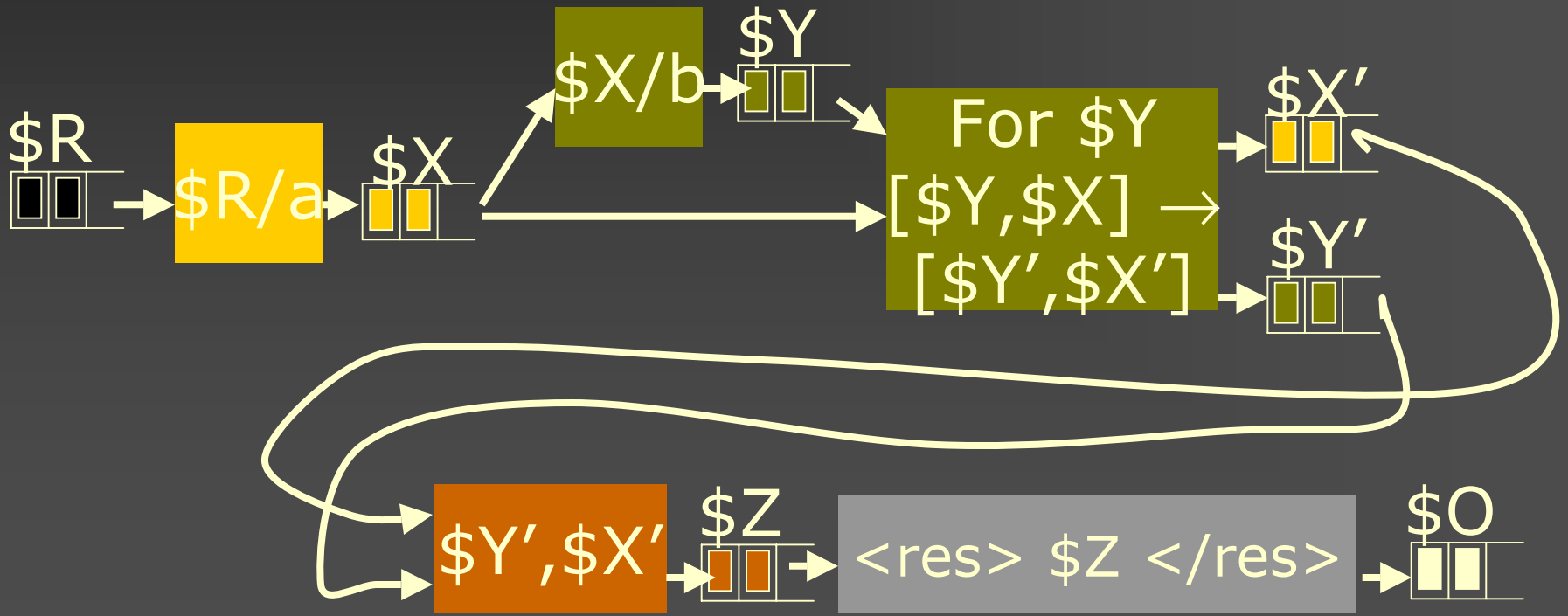
- Motivation
- Architecture
- Framework: Streams + XQuery
- XSM (XML Stream Machine)
- XSM Networks
- Network Composition
- Conclusions

XSM Networks: Intermediate Step in Translating Queries to XSMs

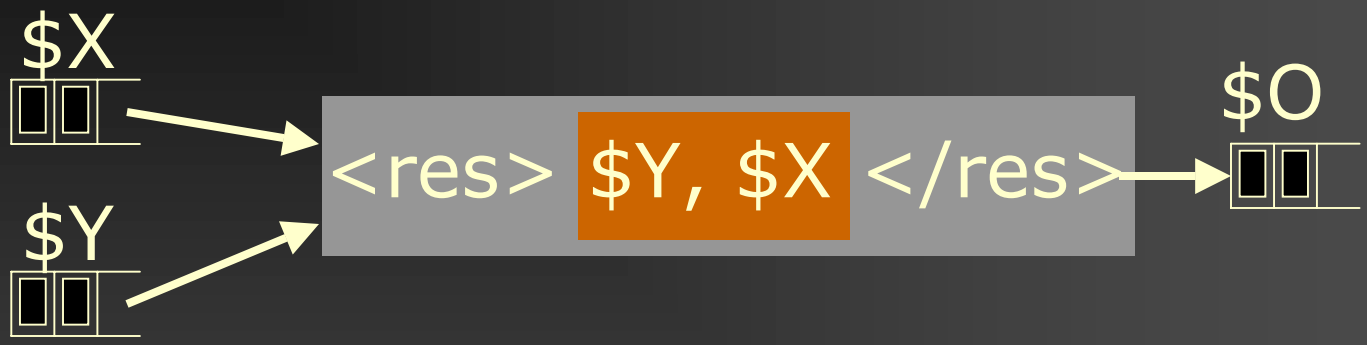


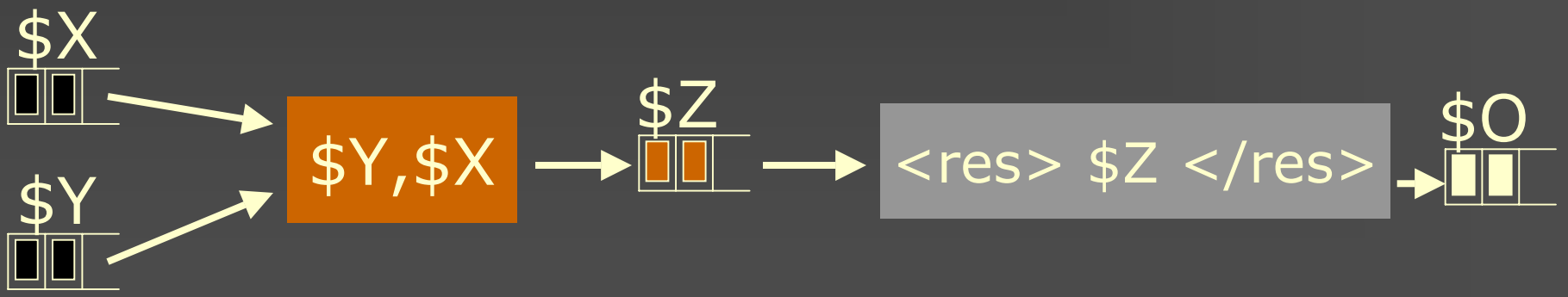
XSM Network

for $X **in** $R/a **return**
  for $Y **in** $X/b **return**
    <res> $Y, $X </res>



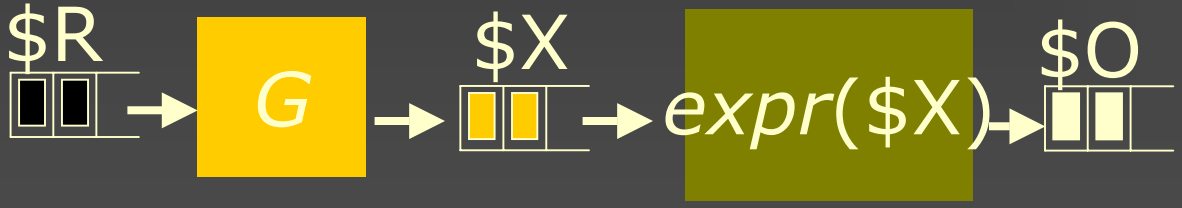
From XQueries to XSM Networks: Non-FLWR Expressions





From XQueries to XSM Networks: FLWRs without Free Variables

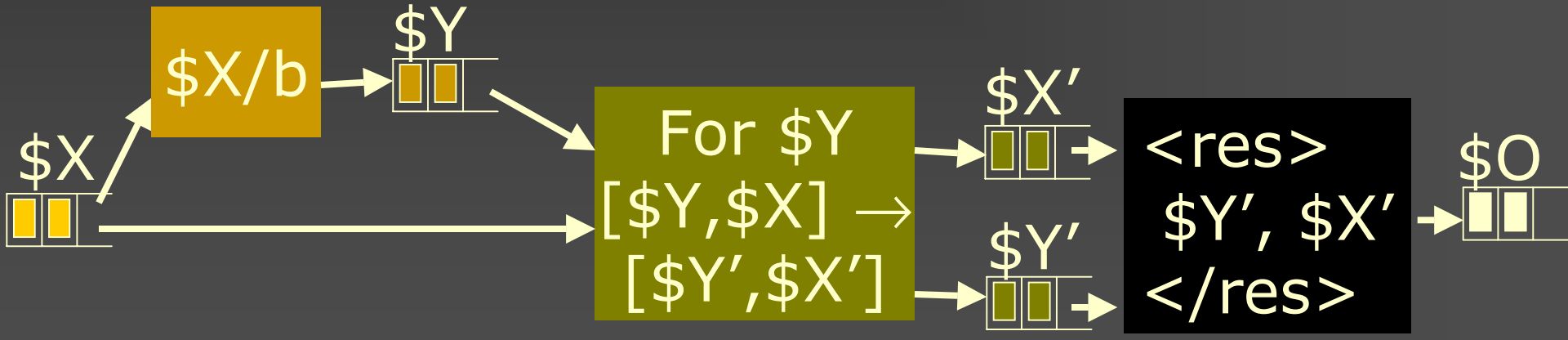
for $X **in** *G* **return**
expr($X)



From XQueries to XSM Networks: FLWRs with Free Variables

```
for $Y in $X/b return
    <res> $Y, $X </res>
```

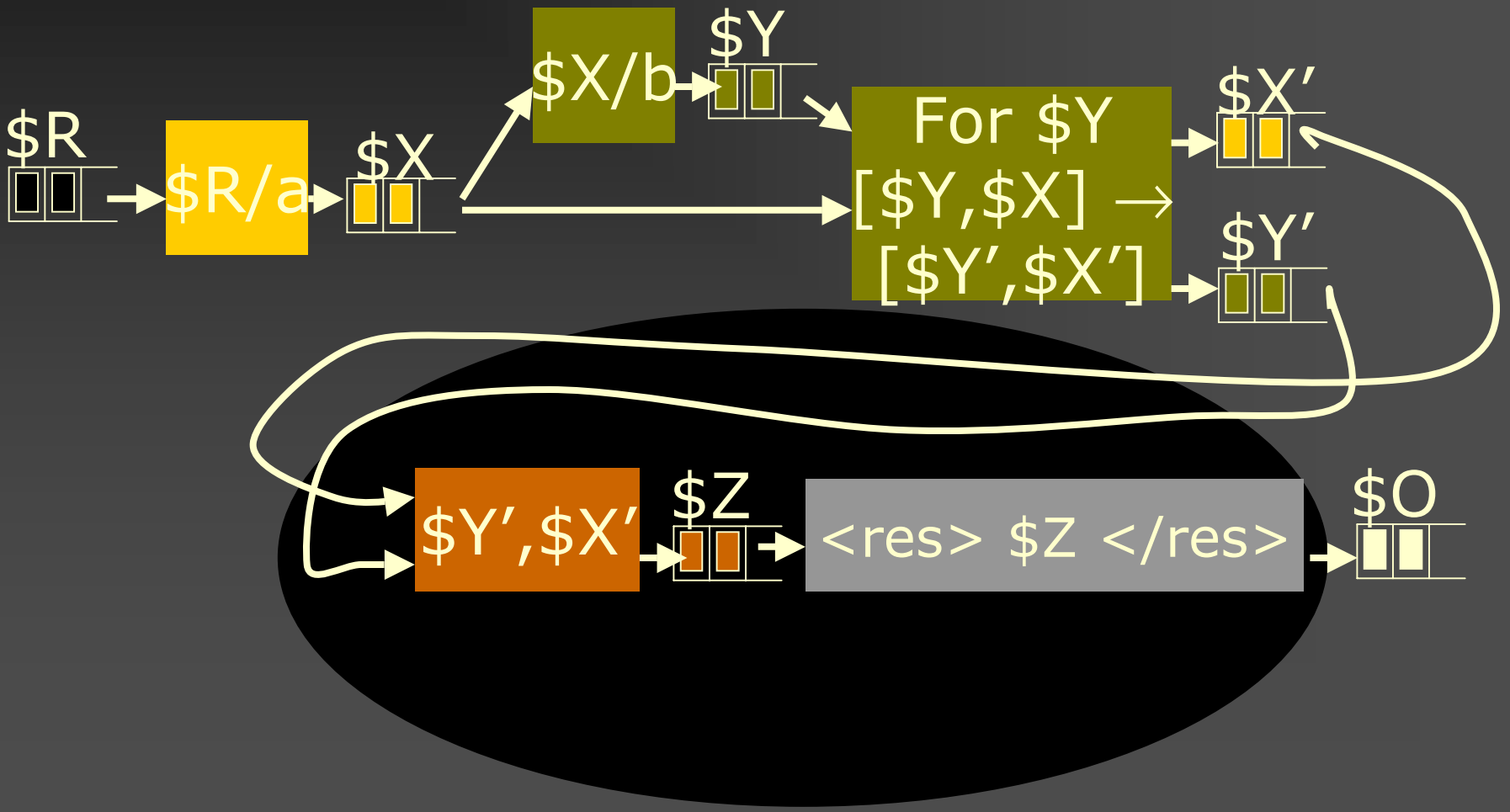
free variable $X



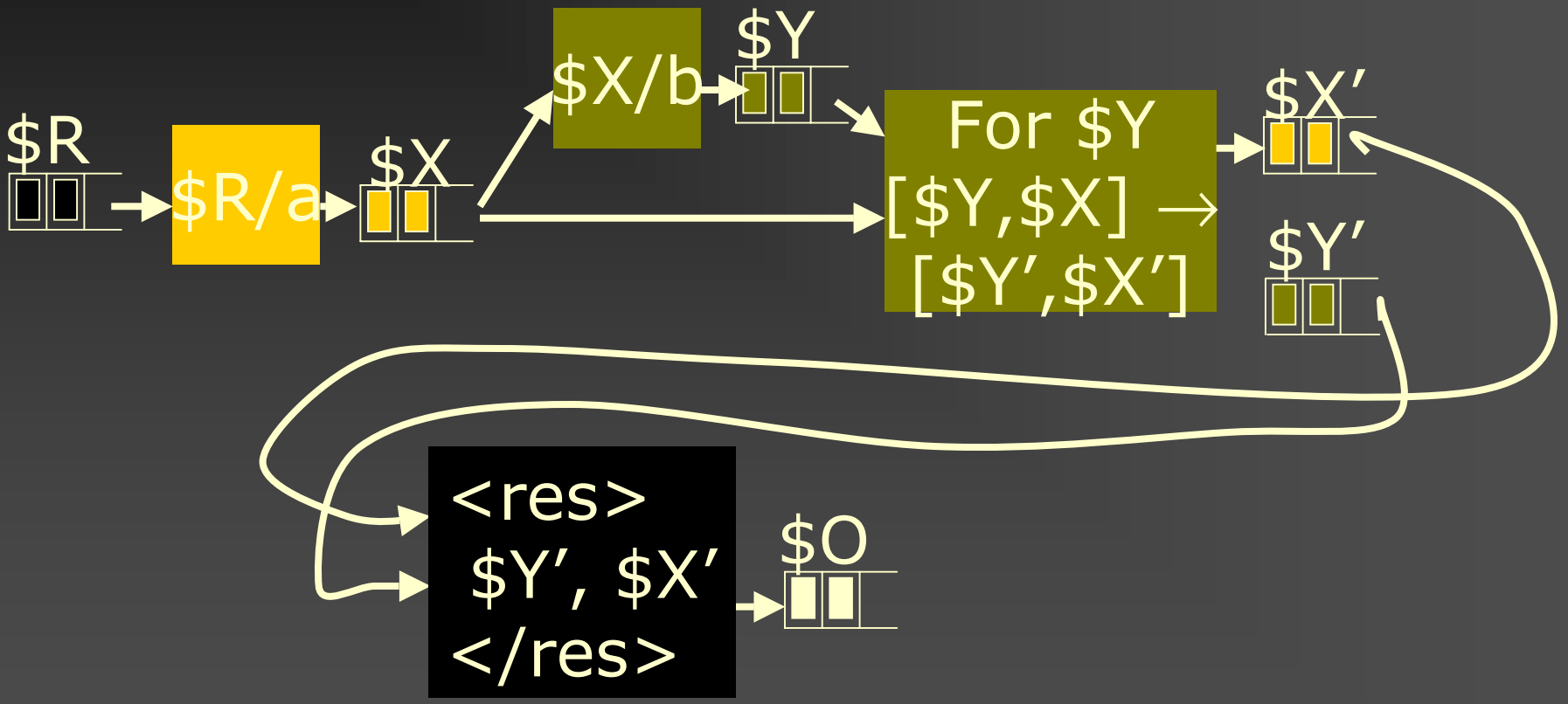
Overview

- Motivation
- Architecture
- Framework: Streams + XQuery
- XSM (XML Stream Machine)
- XSM Networks
- Network Composition
- Conclusions

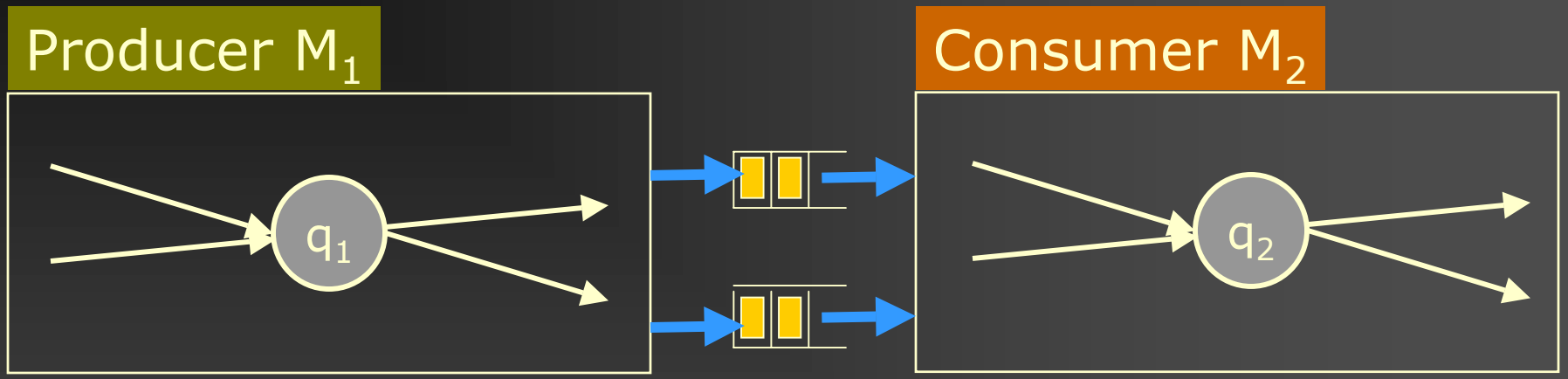
Composition Merges Two XSMs Into One



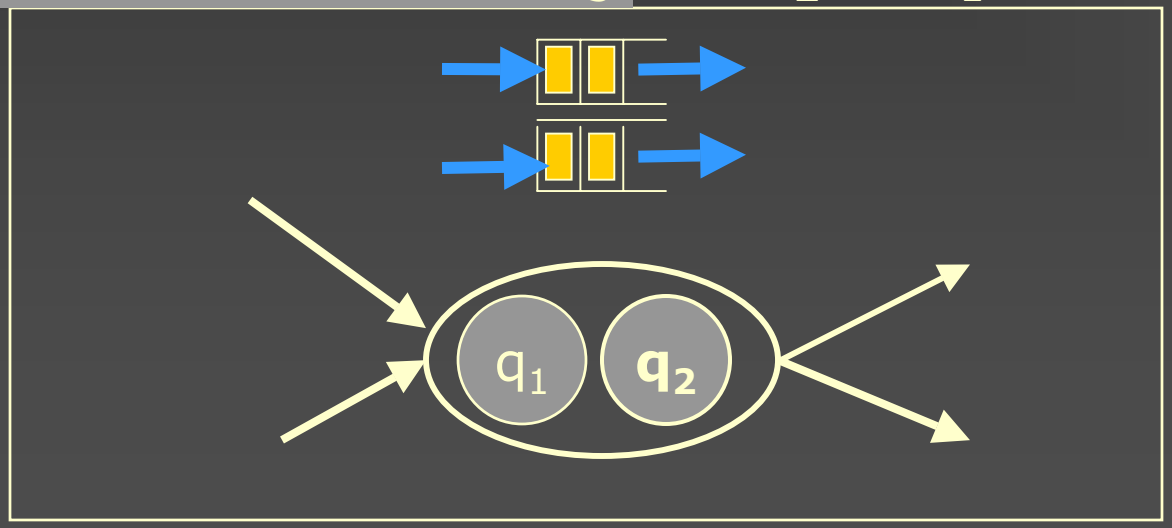
Composition Merges Two XSMs into One



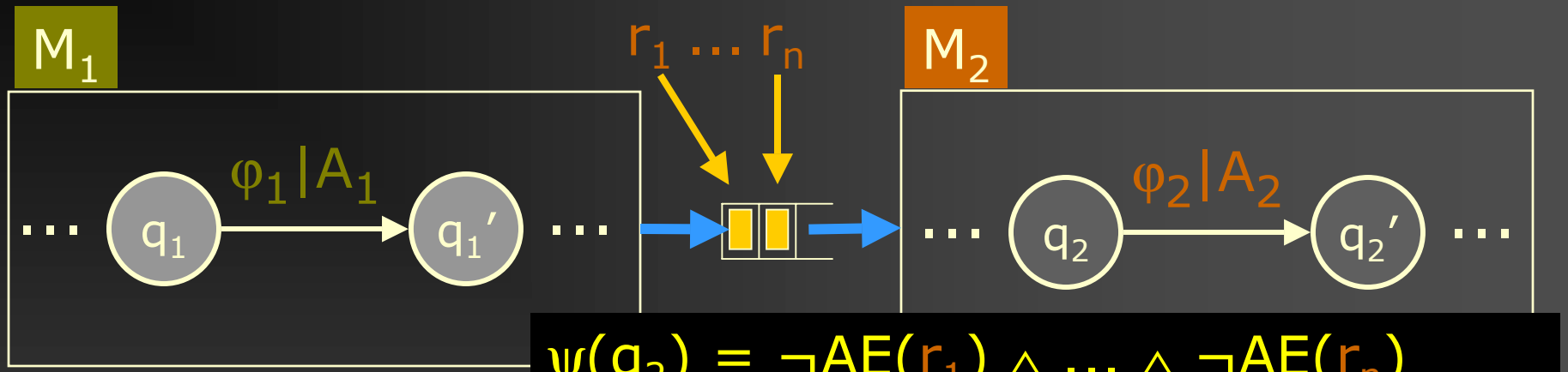
XSM Composition: “State Product” Emulates Producer-Consumer



“State Product” M_3 = (M_2 o M_1)

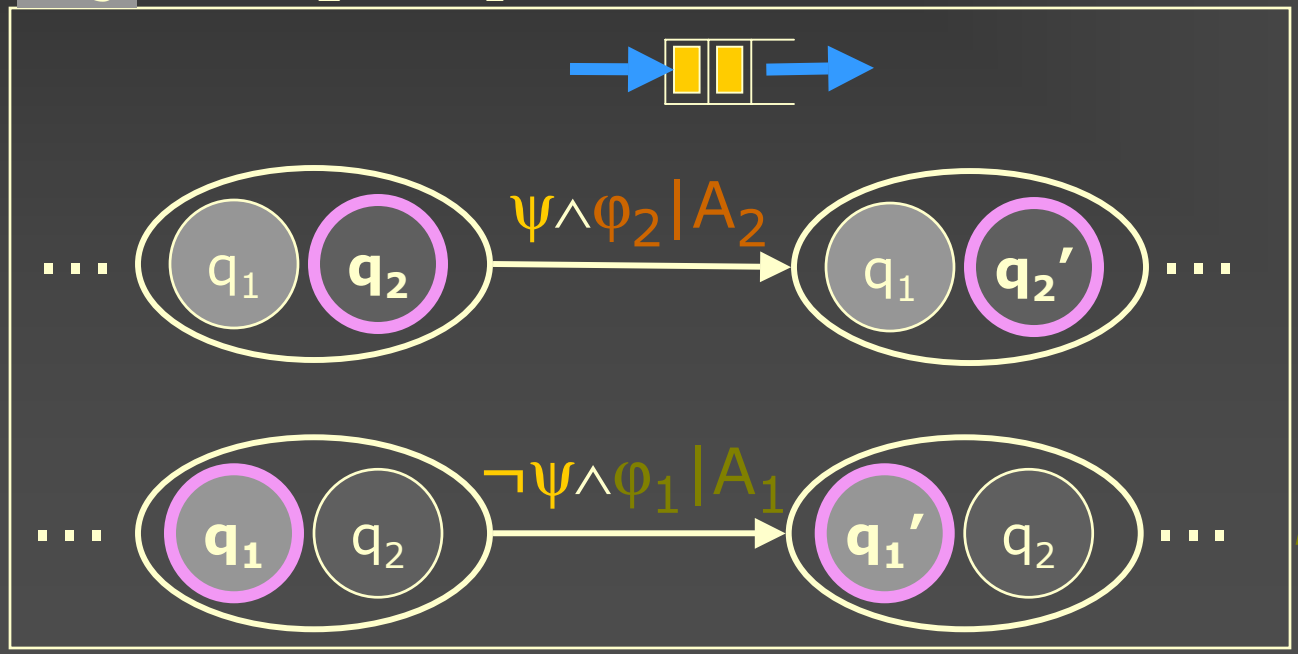


Naive Composition



$\psi(q_2) = \neg AE(r_1) \wedge \ldots \wedge \neg AE(r_n)$
= "no shared read-pointer r_i of q_2 is At End"

$M_3 = (M_2 \circ M_1)$



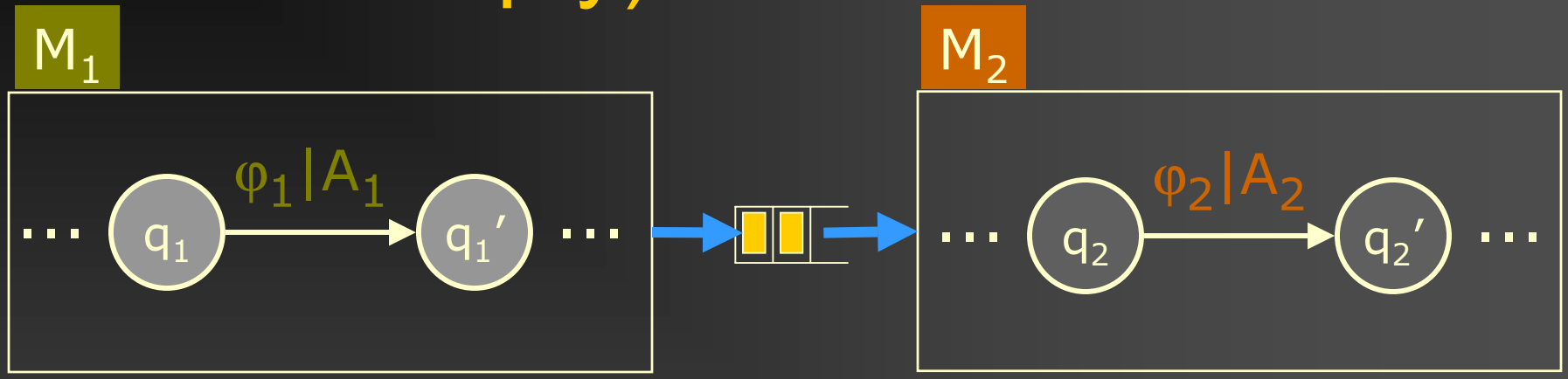
M_2 step if $\psi(q_2)$

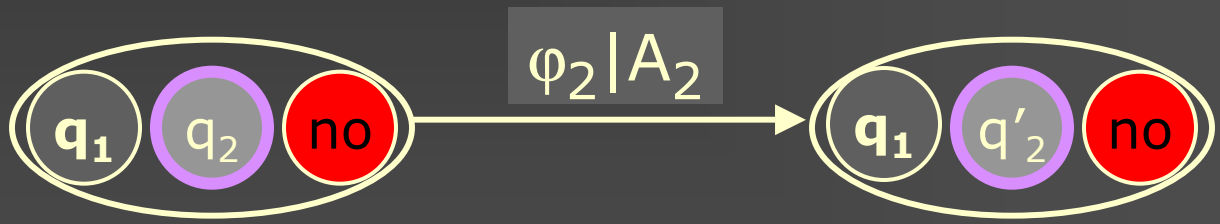
M_1 step if $\neg\psi(q_2)$

Smart Composition

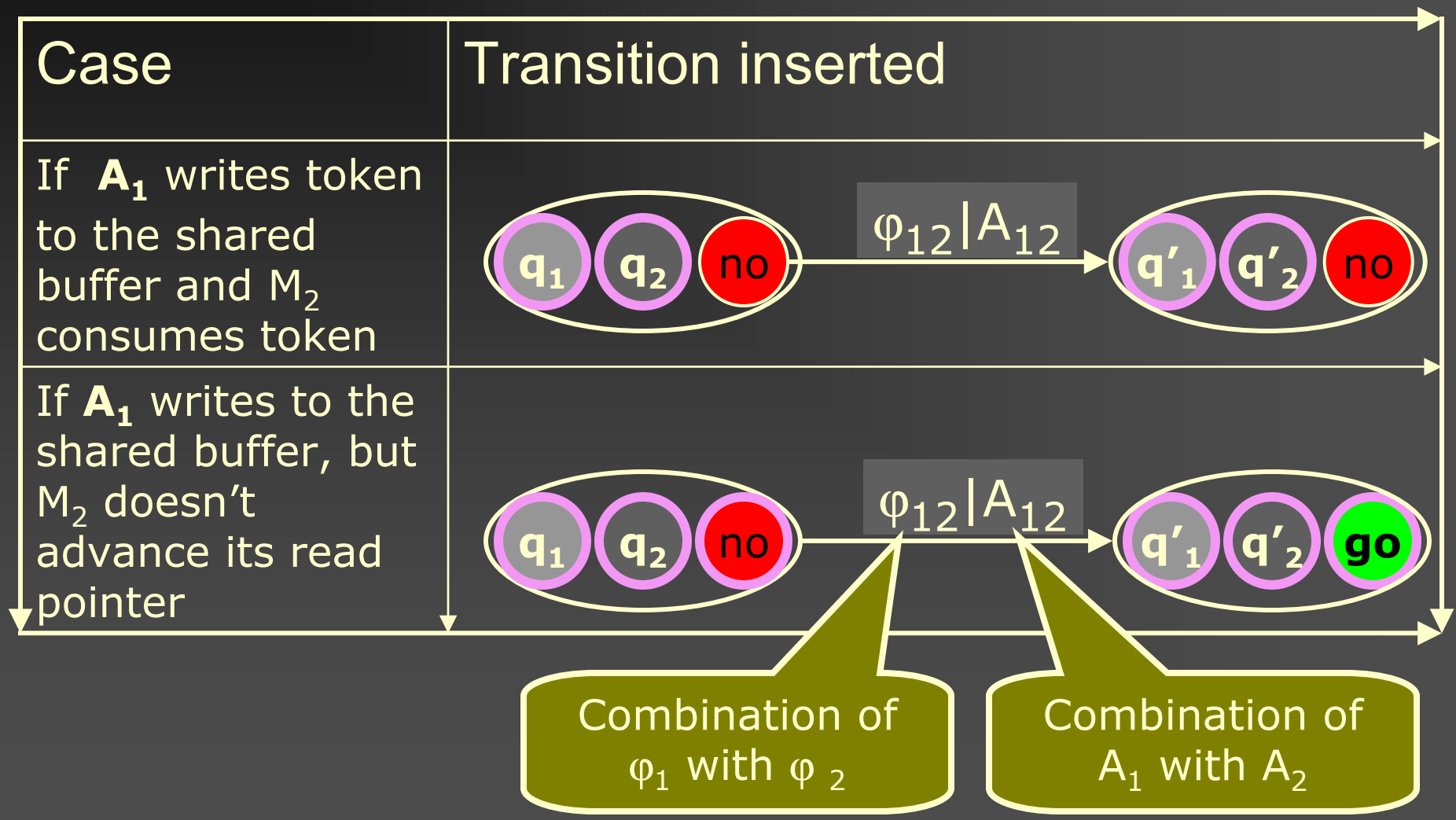
- Normalization Assumptions:
  - #(read-pointers-into-shared-buffer(q_2)) $\leq$ 1
  - Atomic actions only
- Basic idea:
  - avoid runtime tests ("**A**t-**E**nd") whenever outcome can be determined at compile-
- Different "modes":
  - go: consumer M_2 proceeds (full buffer)
  - no: producer M_1 proceeds (empty buffer)
    - may be consumer can follow immediately
  - ae: do runtime check **AE:**

Smart Composition: no Case (shared buffer is empty)



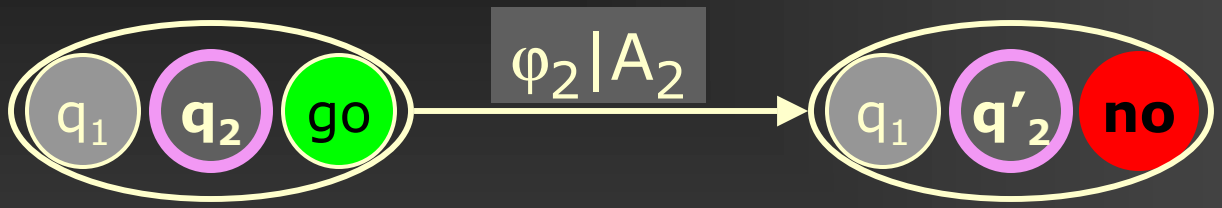
Case	Transition inserted
M_2 does not wait on shared buffer	
A_1 does not write to the shared buffer	

Smart Composition: Producer fills buffer

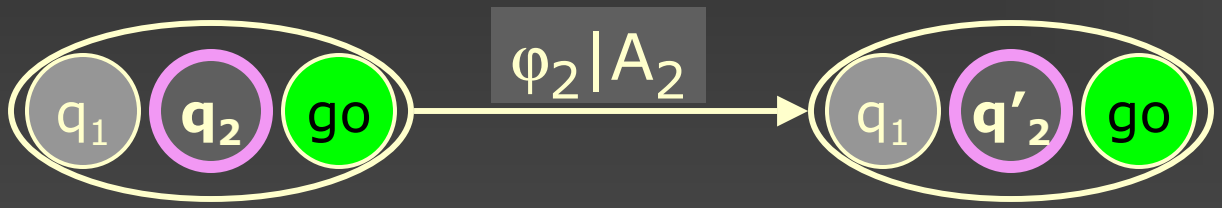


Smart Composition: go - ae - no

- in go mode



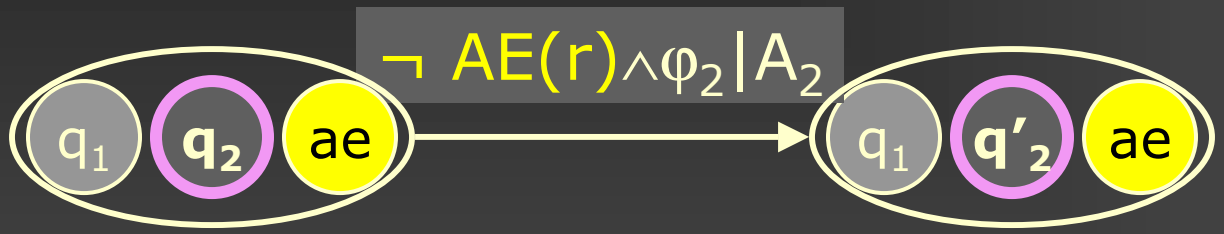
if A_2 advances the read pointer into shared buffer



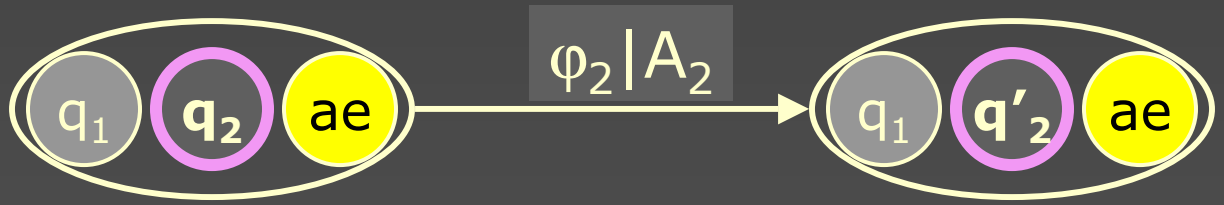
if A_2 does not advance read pointer into shared buffer

Smart Composition: go - ae - no

- in ae mode: insert transitions for M_2 step if possible ...



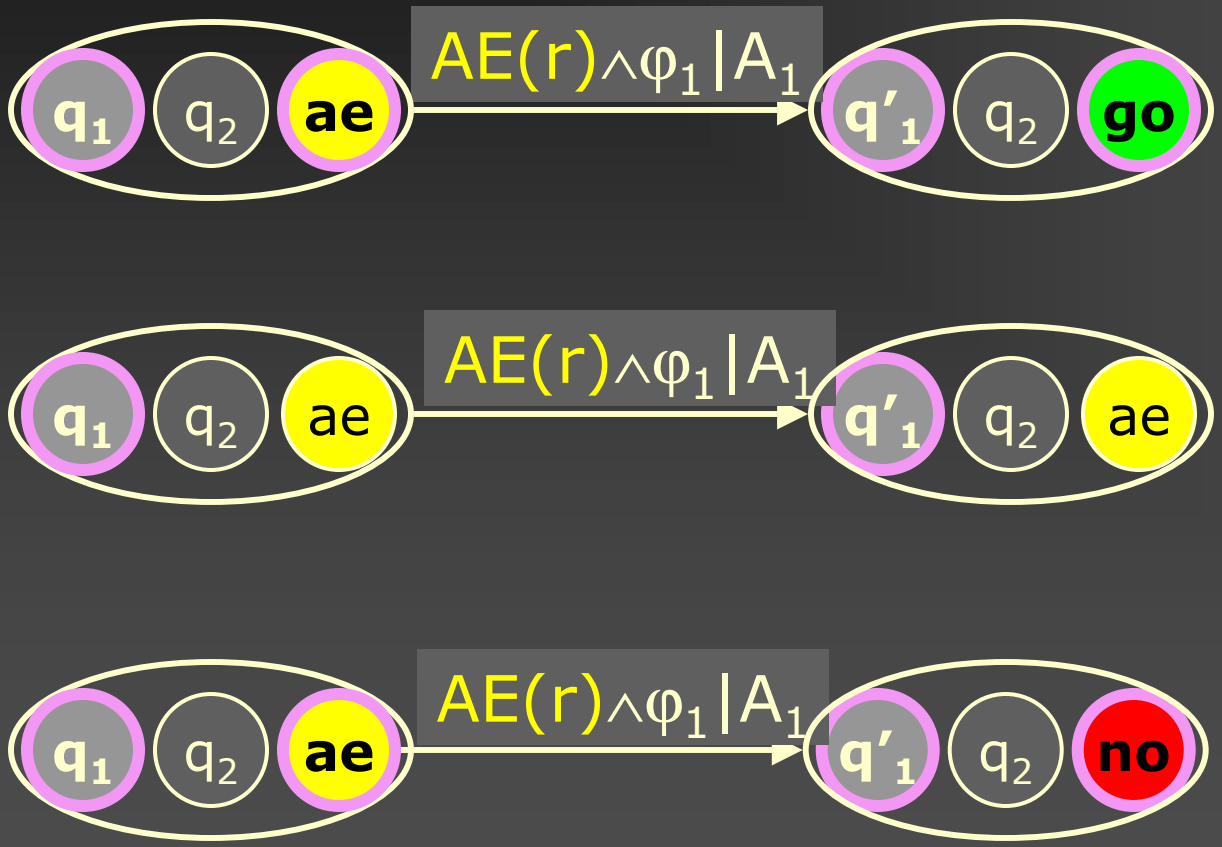
if $\varnothing_2$; A_2 has a read from the shared buffer



If $\varnothing_2$; A_2 has no read from the shared buffer

Smart Composition: go - ae - no

- AND transitions corresponding to M_1 step ...



if A_1 has one write into the shared buffer

if A_1 has more than one write into the shared buffer

if A_1 has no write into the shared buffer

Performance Datapoint (Transformation Query on DBLP)

Data Size (KB)	Xalan (ms)	XSM Java	XSM C
4	663	266	30
5000	7031	2360	312
20000	102710	8266	1156
80000		32078	4640

Conclusions & Future Work

- Novel query processor model
- Success in filtering & transformation

- To be extended for joins & aggregations
- Memory footprint questions
  - Facilitated by model's simplicity

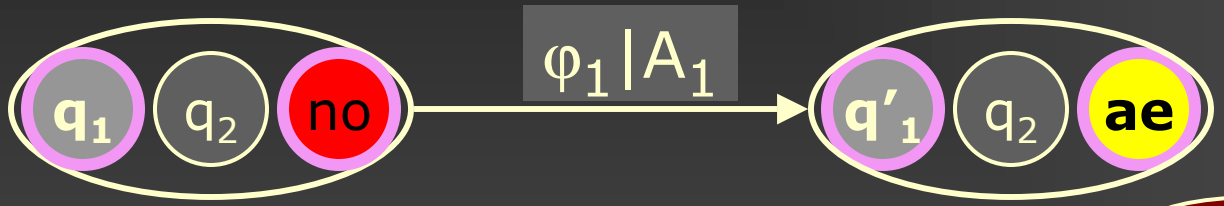
Related Work

- Relational Data Streams & Sequence Data Models
- Pipelined Join Operators
- Aggregates & Approximations
- Fast XPath on streams
- Memory requirements of validating XML

Smart Composition: go - ae - no

- in no mode: execute M_1 step ...

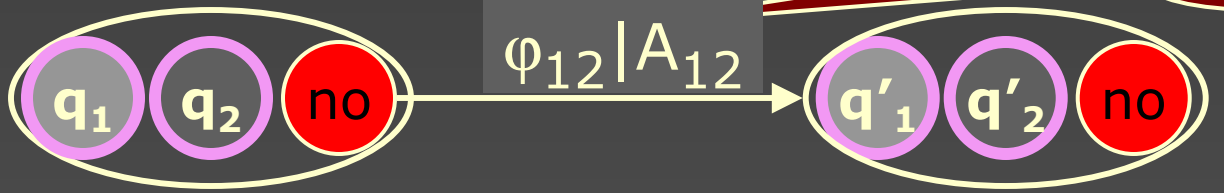
if A_1 does not advance shared write pointer



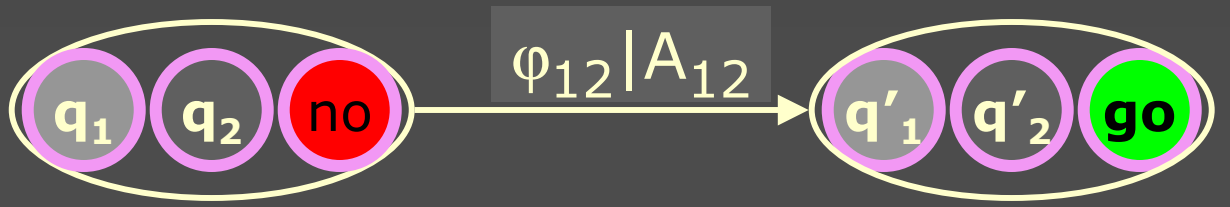
if A_1 does advance shared write pointer

- ... AND possibly M_2 step

simplified composed $\varphi_1 \wedge \varphi_2$ and $(A_1;A_2)$



if A_2 advances shared read pointer



if A_2 does not advance shared read pointer