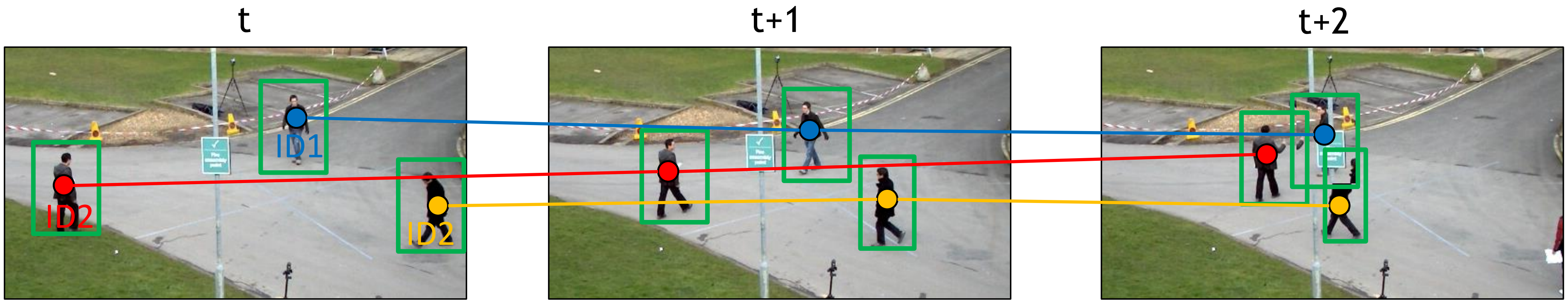
Advanced CV methods
Multiple object tracking

Matej Kristan

Laboratorij za Umetne Vizualne Spoznavne Sisteme,
Fakulteta za računalništvo in informatiko,
Univerza v Ljubljani

Multiple Object Tracking (MOT) task

- Typically assumes a pretrained object detector available (e.g., pedestrians)
- Analyze detections to recover trajectories for "all" specified class of objects



- Assume that objects may:

  (i) enter the scene, (ii) leave the scene or get occluded, (iii) re-enter the scene
- The number of objects (in general) unknown in advance

Online vs Batch Multiple Object Tracking

- Batch tracking considers “all” frames to infer position at t

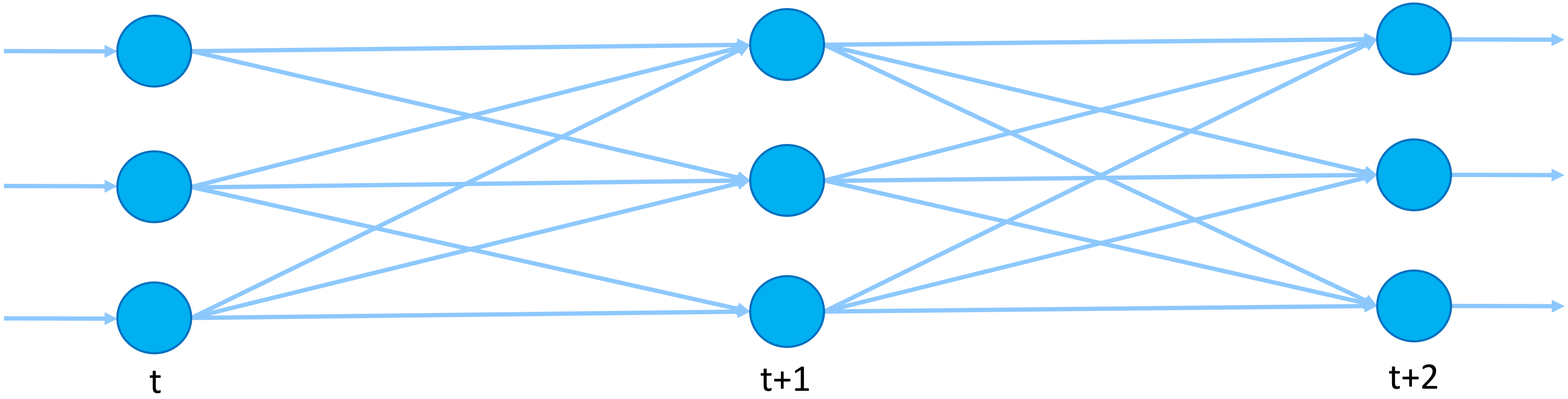
  (useful for offline applications, e.g., post-hoc analysis, video editing)

- Online tracking consider only frames before *t* to infer position at *t*

  (useful in realtime applications, e.g., drones)

Batch tracking: Network flow formulation

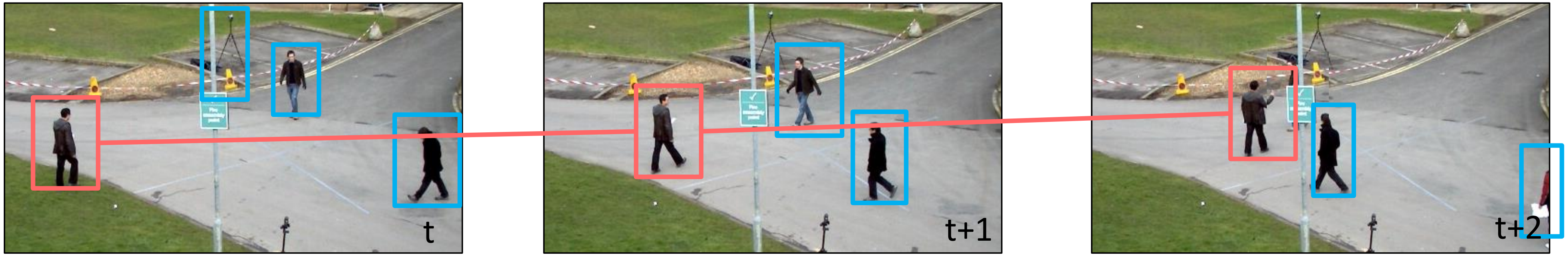
- Tracking: finding paths through a graph constructed over a sequence of detections *(detection=node, between-frame track=vertex)*

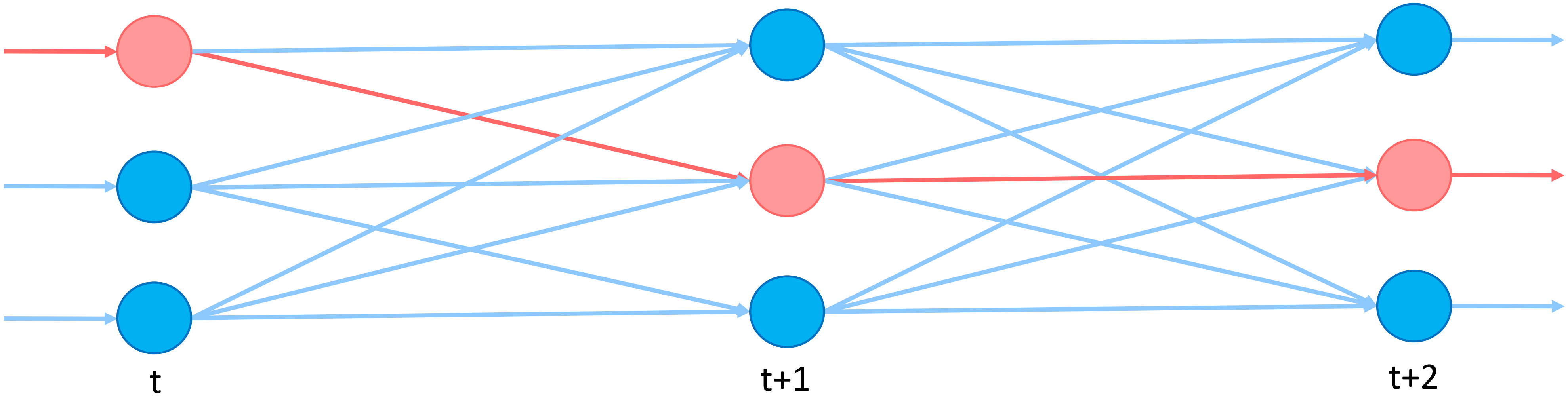


L. Leal-Taixé et al. "Everybody needs somebody: Modeling social and grouping behavior on a linear programming multiple people tracker." ICCVW2011

Batch tracking: Network flow formulation

- Tracking: finding paths through a graph constructed over a sequence of detections *(detection=node, between-frame track=vertex)*

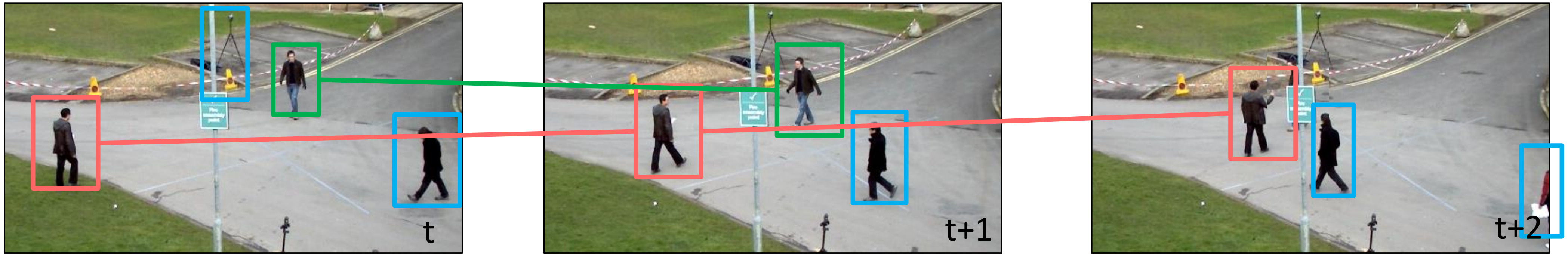


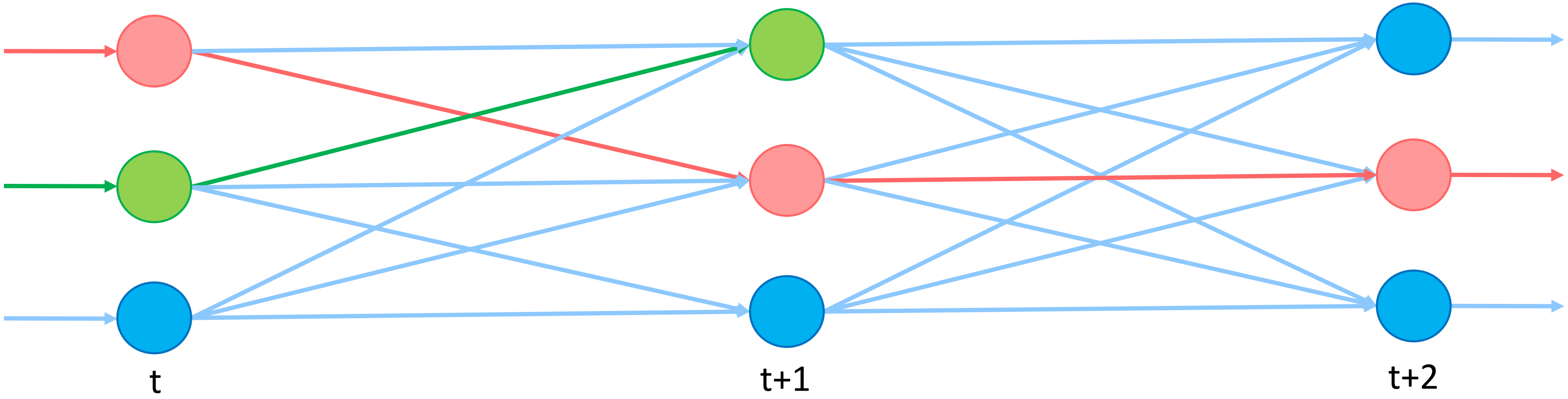


L. Leal-Taixé et al. "Everybody needs somebody: Modeling social and grouping behavior on a linear programming multiple people tracker." ICCVW2011

Batch tracking: Network flow formulation

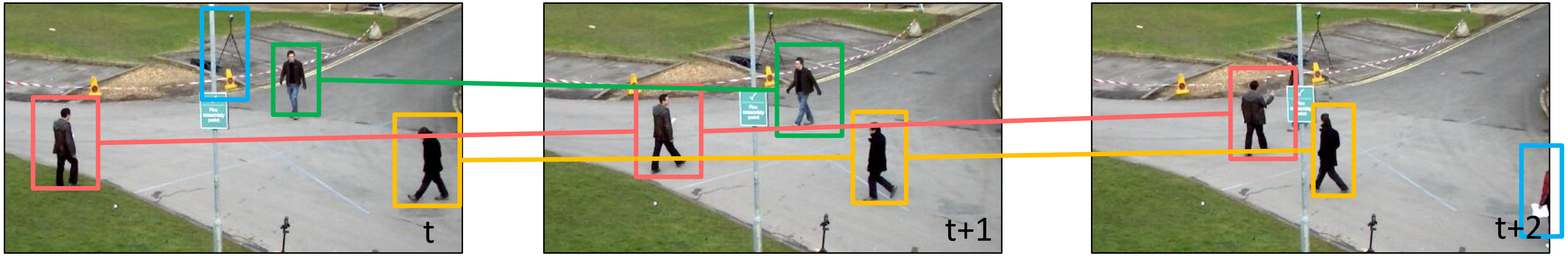
- Tracking: finding paths through a graph constructed over a sequence of detections *(detection=node, between-frame track=vertex)*

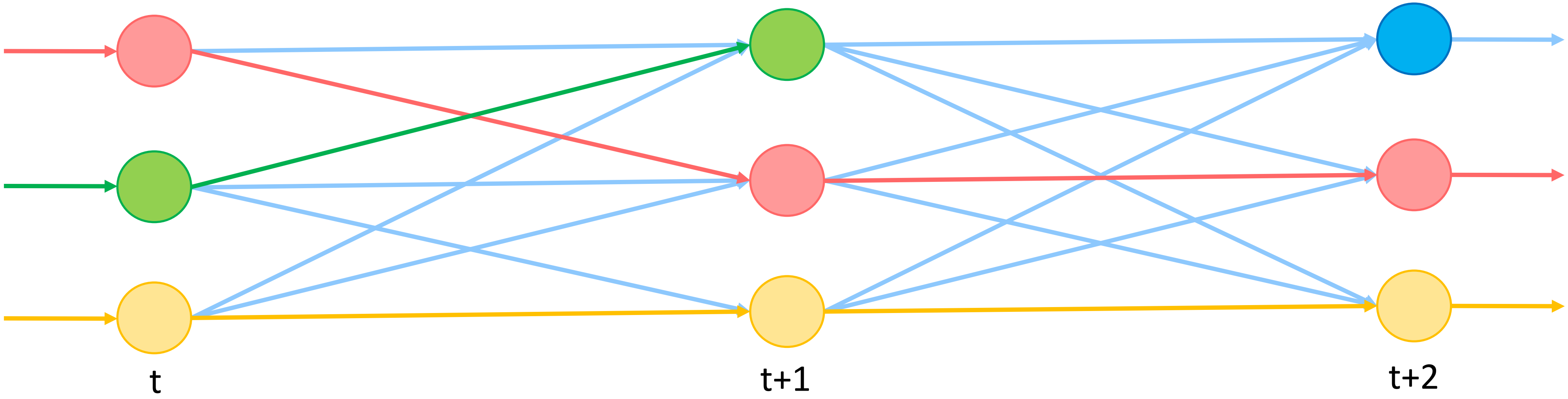




Batch tracking: Network flow formulation

- Tracking: finding paths through a graph constructed over a sequence of detections *(detection=node, between-frame track=vertex)*

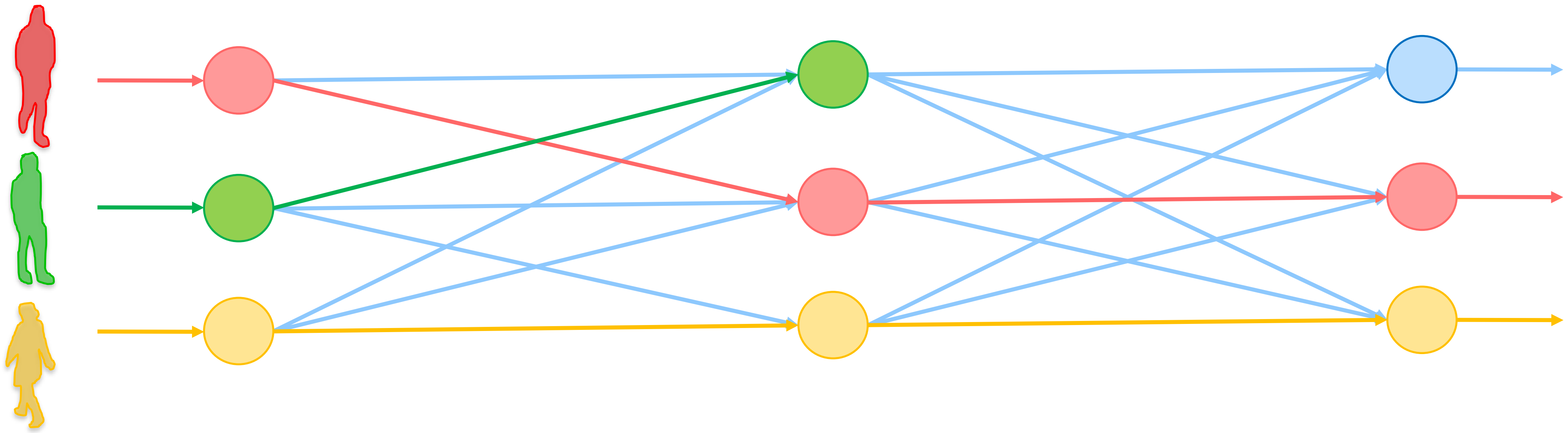




Network flow formulation

- Solve the Minumum Cost Flow problem:

  "Determine the minimum cost of shipment of a commodity through a network"



Node = detection ; Edge = flow = trajectory ; 1 unit of flow = target

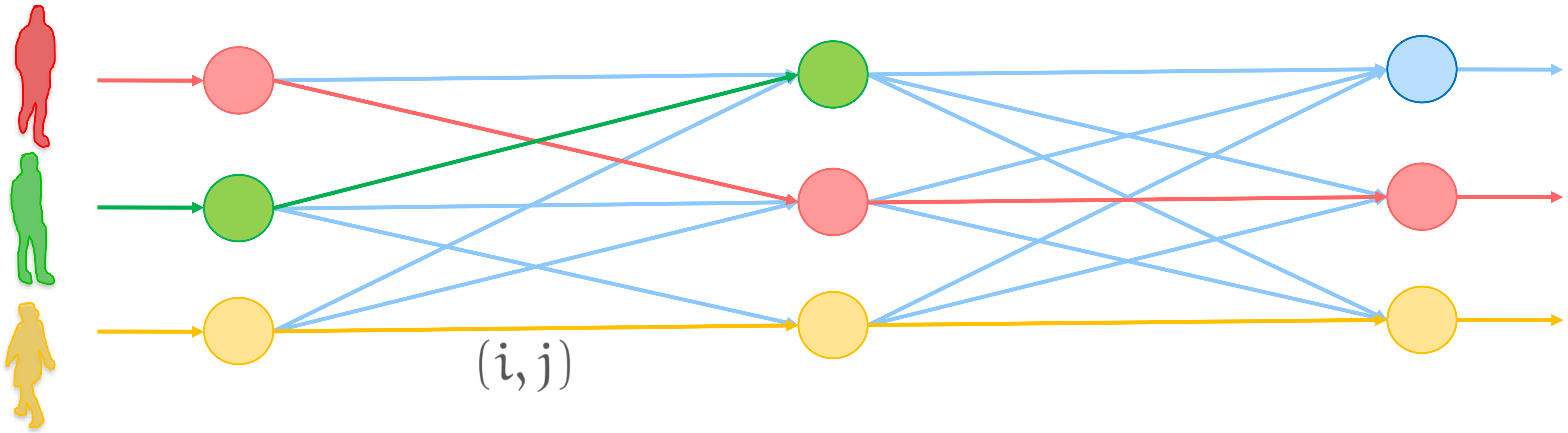
Find a set of trajectories $\mathcal{T}$ *that minimize the flow cost*

L. Leal-Taixé et al. "Everybody needs somebody: Modeling social and grouping behavior on a linear programming multiple people tracker." ICCVW2011

Ravindra K. Ahuja; Thomas L. Magnanti & James B. Orlin. "Network Flows: Theory, Algorithms, and Applications". Prentice-Hall, Inc. 1993

Network flow formulation

- The cost function: $\mathcal{T}* = \arg\min_{\mathcal{T}} \sum_{i,j} C(i,j) f(i,j)$

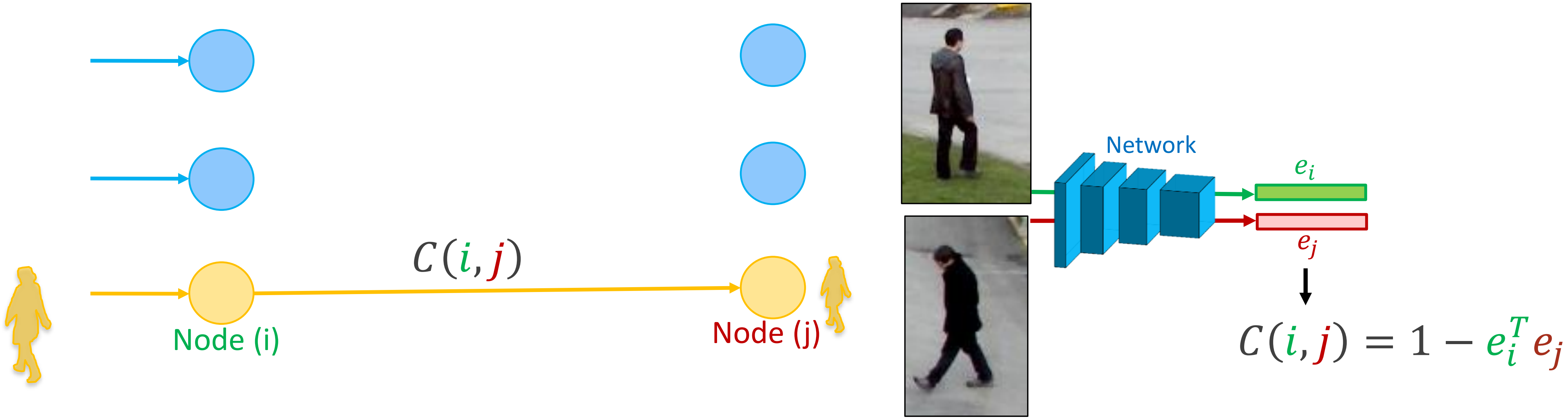


- To each edge (i, j) assign:
  - A cost $C(i, j) \in \mathbb{R}$ and activation indicator $f(i, j) \in \{0,1\}$

L. Leal-Taixé et al. "Everybody needs somebody: Modeling social and grouping behavior on a linear programming multiple people tracker." ICCVW2011

Network flow formulation

- Edge cost may be visual similarity between two detections
- E.g.: A dot product between descriptors e extracted at detections i and j

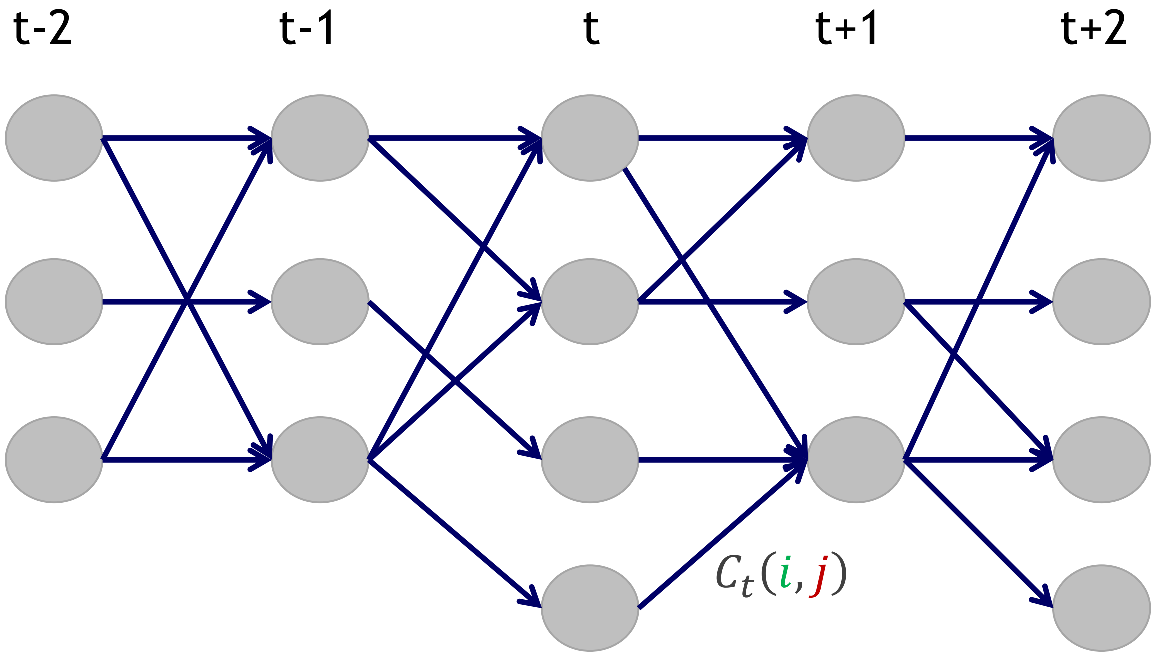


- High similarity = low cost -> $C(i,j) \downarrow$
- Low similarity = high cost -> $C(i,j) \uparrow$

L. Leal-Taixé et al. "Everybody needs somebody: Modeling social and grouping behavior on a linear programming multiple people tracker." ICCVW2011

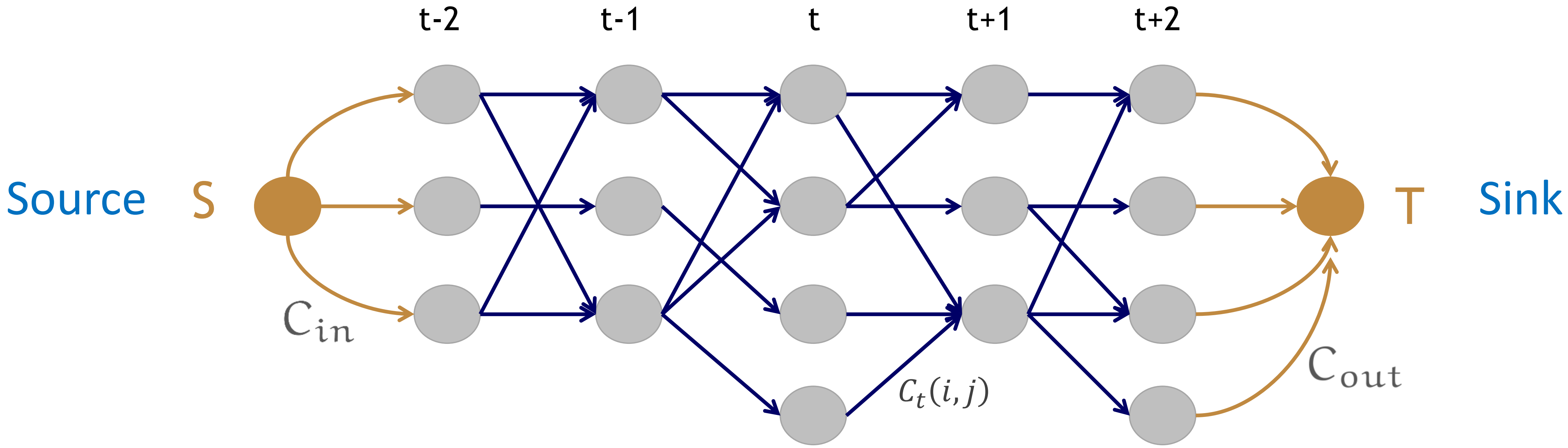
Ravindra K. Ahuja; Thomas L. Magnanti & James B. Orlin. "Network Flows: Theory, Algorithms, and Applications". Prentice-Hall, Inc. 1993

Network flow formulation



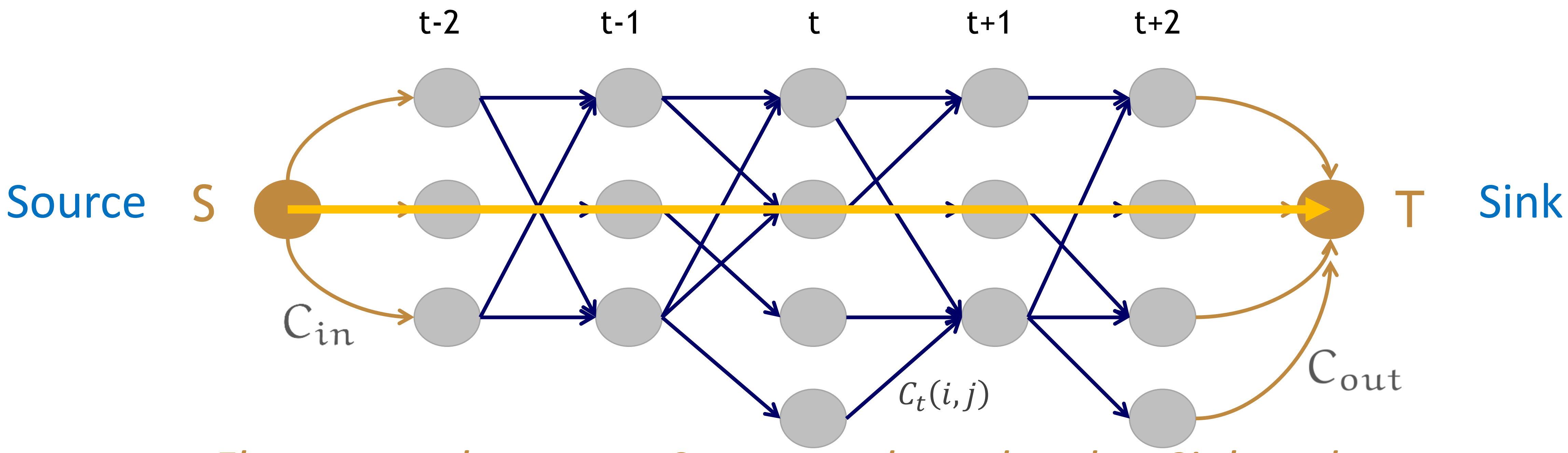
- Flow = Trajectory = Object presence
- Transition cost $C_t(i, j)$: appearance difference between detections

Network flow formulation



- Flow = Trajectory = Object presence
- Transition cost $C_t(i,j)$: appearance difference between detections
- Entrance/exit $C_{in}(i)/C_{out}(i)$: cost to start and end a trajectory

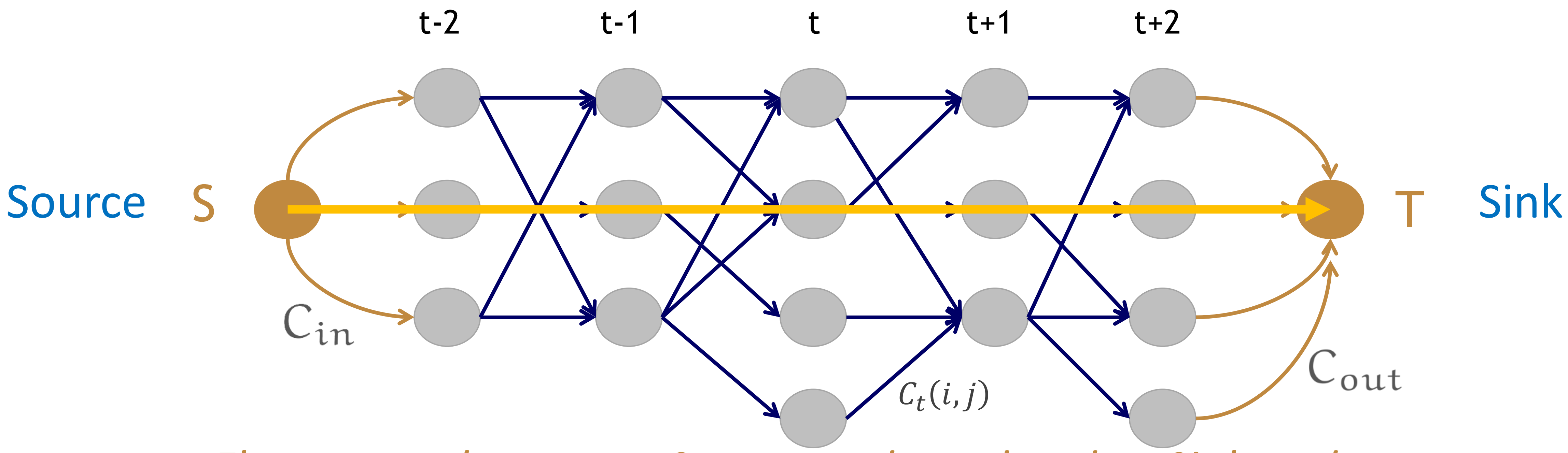
Network flow formulation



Flow can only start at Source node and end at Sink node.

- Flow = Trajectory = Object presence
- Transition cost $C_t(i, j)$: appearance difference between detections
- Entrance/exit $C_{in}(i)/C_{out}(i)$: cost to start and end a trajectory

Network flow formulation



Flow can only start at Source node and end at Sink node.

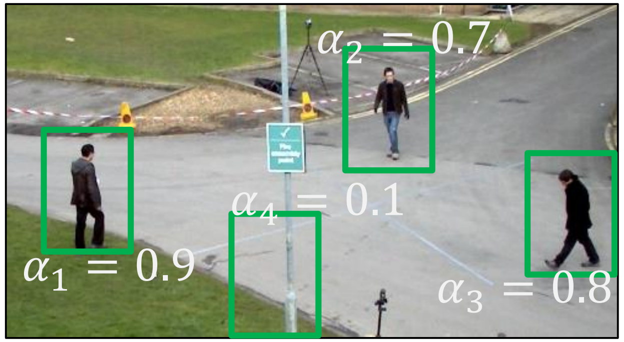
- Transition cost $C_t(i,j)$: appearance difference between detections
- Entrance/exit $C_{in}(i)/C_{out}(i)$: cost to start and end a trajectory

$$\mathcal{T}* = \arg\min_{\mathcal{T}} \sum_{i,j} C(i,j) f(i,j)$$

BUT, since all costs are >0, the trivial solution is $f(i,j) = 0$, for all (i,j)!

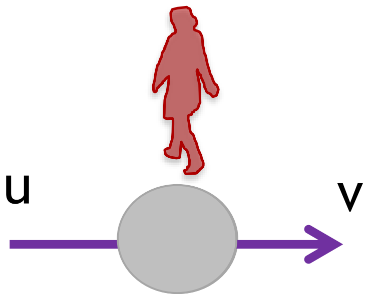
Network flow formulation

- Introduce a negative cost that reflects the quality of the detection

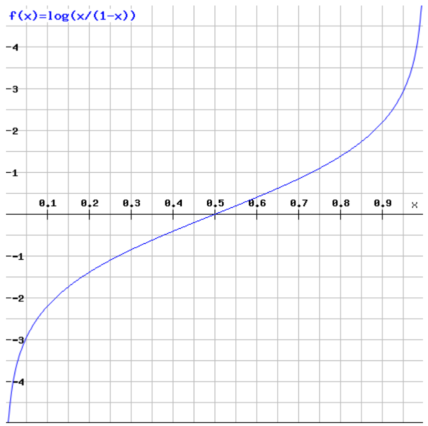


$\beta_i = 1 - \alpha_i$ … probability that a detection *(i)* is a false alarm

- Augment the graph by a "detection edge" for each node

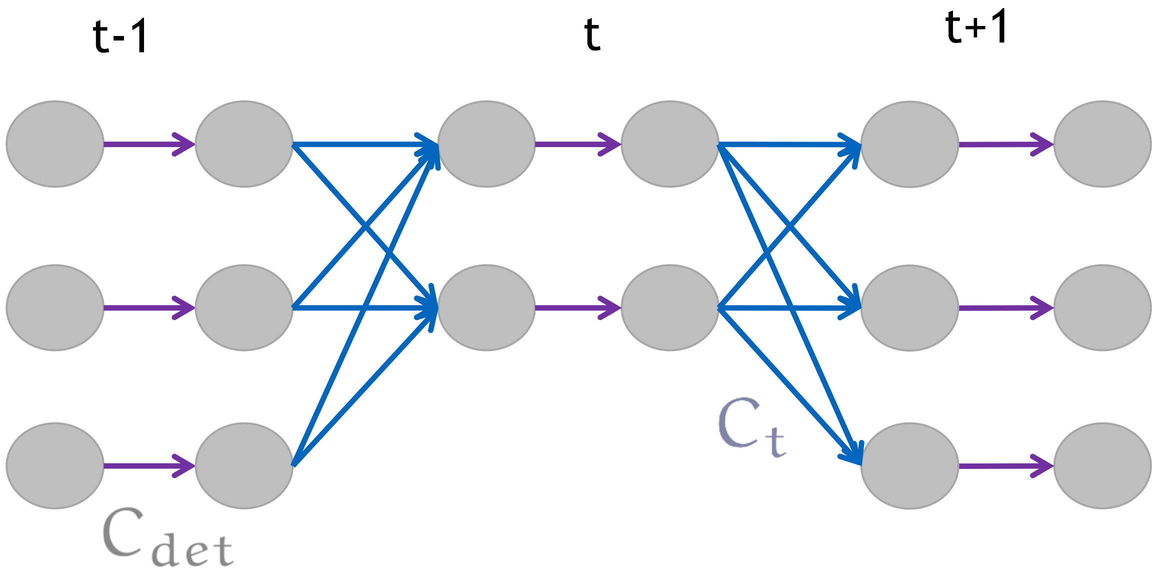


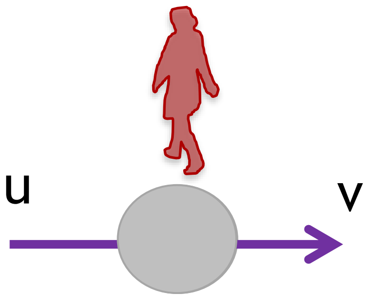
$$C_{det} = \log \frac{\beta_i}{1-\beta_i}$$



Network flow formulation

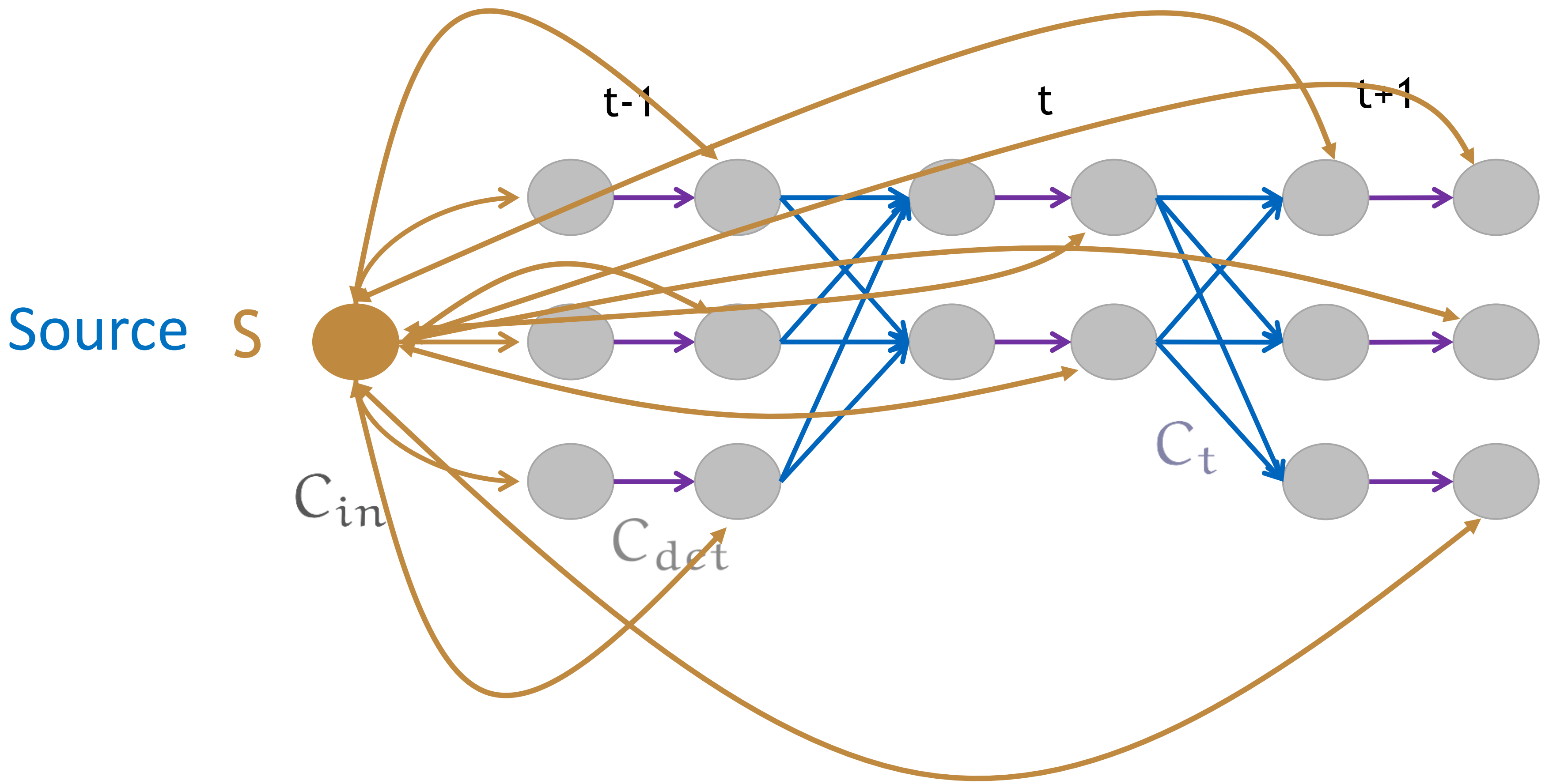
- The complete graph with the detection edges inserted looks like this:





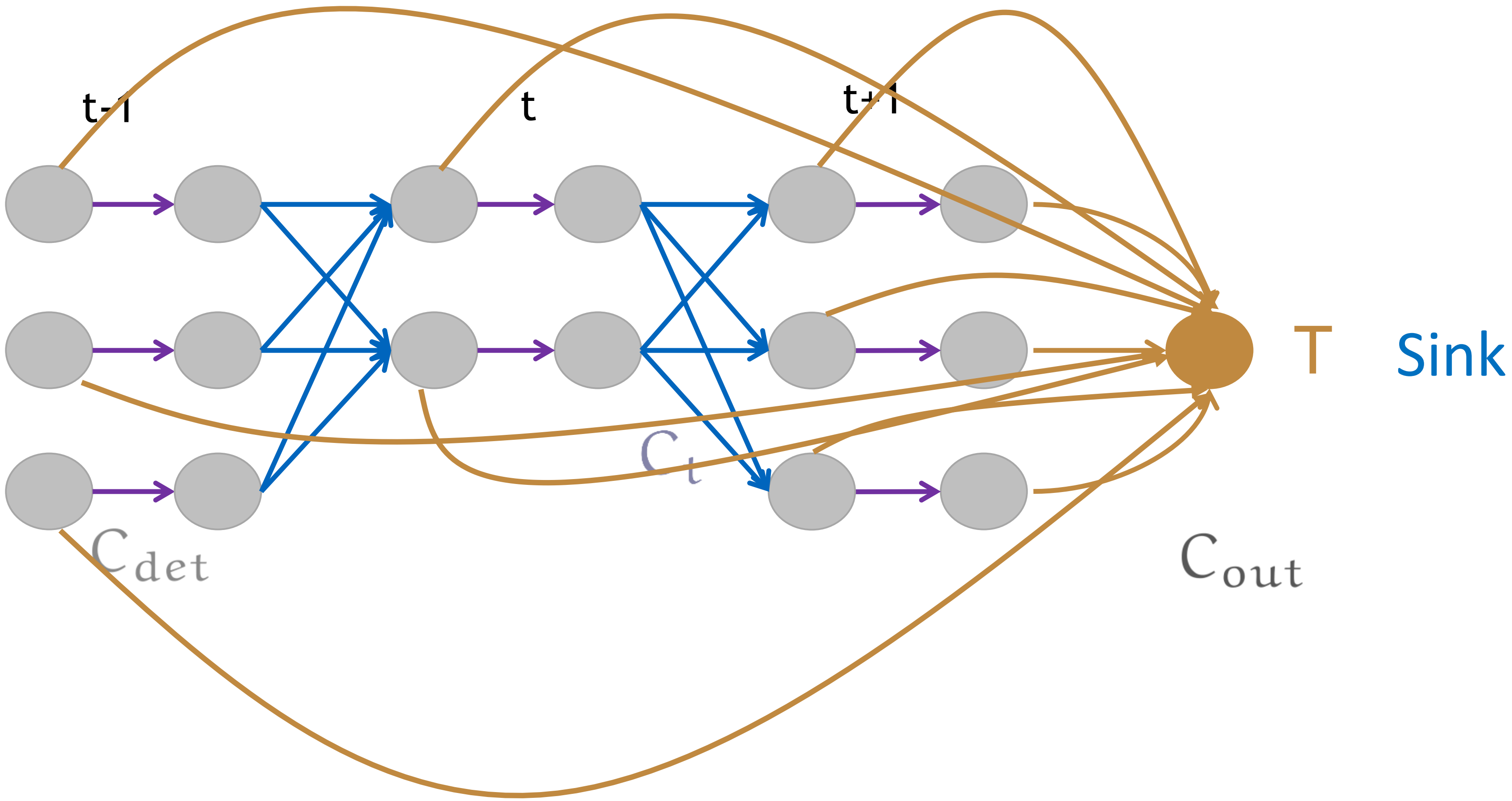
Network flow formulation

- Add connections that allow start of trajectory at “every” detection

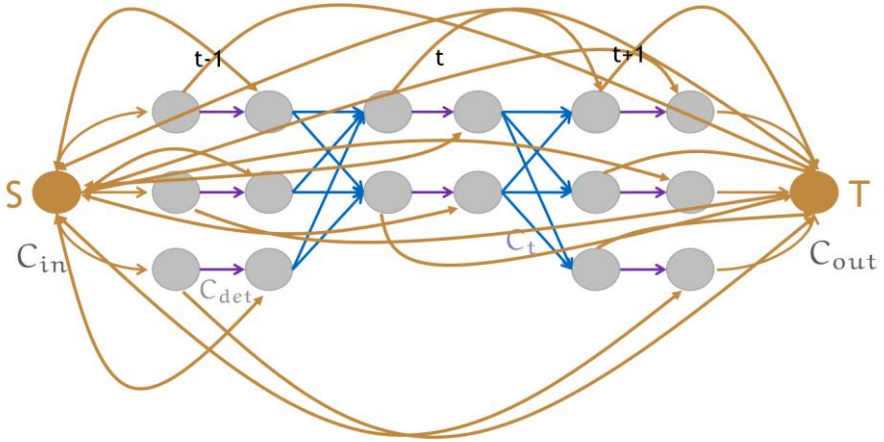


Network flow formulation

- Add connections that allow ending of a trajectory at “every” detection



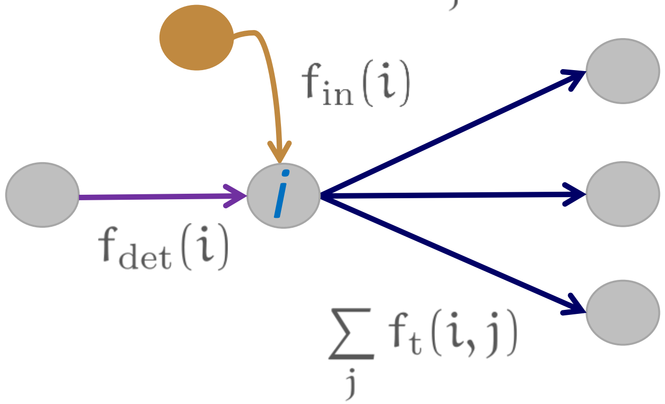
Network flow formulation



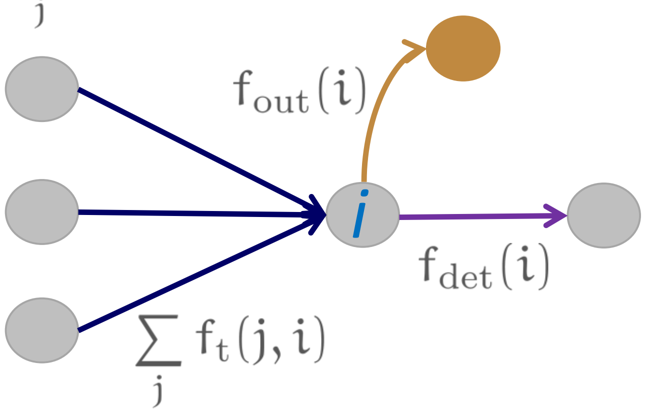
- Objective: $\mathcal{T}* = \arg\min_{\mathcal{T}} \sum_{i,j} C(i,j) f(i,j)$
- Constraints:

1. Flow conservation at all nodes (all flow that goes into a node, goes out)

$$f_{in}(i) + f_{det}(i) = \sum_j f_t(i,j)$$



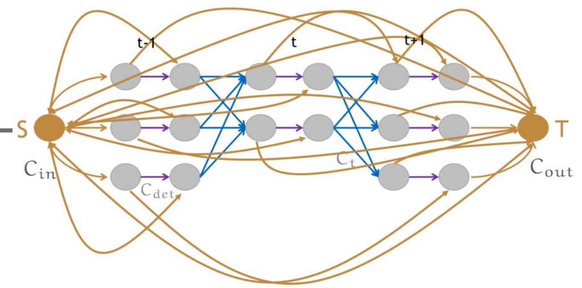
$$\sum_j f_t(j,i) = f_{out}(i) + f_{det}(i)$$



2. Edge capacity constraints:

$$f_{in}(i) + f_{det}(i) \in \{0, 1\} \qquad f_{out}(i) + f_{det}(i) \in \{0, 1\} \qquad f \in \{0, 1\}$$

Network flow formulation



- The model equations:

$$\mathcal{T}* = \arg\min_{\mathcal{T}} \sum_{i,j} C(i,j) f(i,j)$$

$$f_{out}(i) + f_{det}(i) - \sum_j f_t(j,i) = 0$$

$$f_{in}(i) + f_{det}(i) - \sum_j f_t(j,i) = 0$$

$$f_{in}(i) + f_{det}(i) \in \{0,1\}$$

$$f_{out}(i) + f_{det}(i) \in \{0,1\}$$

$$f \in \{0,1\}$$

Summarize algebraically:

$$\boldsymbol{x} = [f_t(i,j) \ldots, f_{in}(i), \ldots, f_{out}(i), \ldots, f_{det}(i), \ldots]^T$$

$$\boldsymbol{c} = [C(i,j) \ldots]^T$$

$$\mathcal{T}_* = \operatorname*{argmin}_{\mathcal{T}} \boldsymbol{c}^T \boldsymbol{x}$$

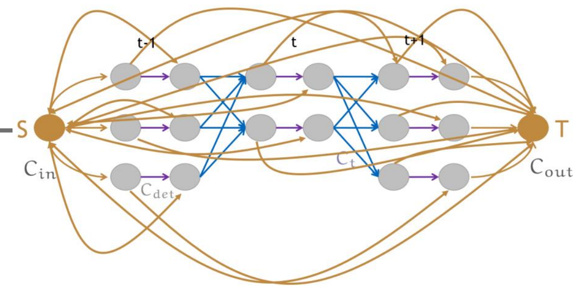
$$\boldsymbol{Ax} = \boldsymbol{0} \qquad , \boldsymbol{A} = \begin{bmatrix} 1,1,\ldots,-1,-1\ldots \\ \ldots \end{bmatrix}$$

$$\boldsymbol{A_2 x} \in [\boldsymbol{0},\boldsymbol{1}] \qquad , \boldsymbol{A_2} = \begin{bmatrix} 1,1,0,\ldots,0 \\ \ldots \end{bmatrix}$$

$$\boldsymbol{x} \in [\boldsymbol{0},\boldsymbol{1}]$$

Binary optimization – NP-hard!!

Network flow formulation



- The model equations:

$$\mathcal{T}* = \arg\min_{\mathcal{T}} \sum_{i,j} C(i,j) f(i,j)$$

------- Constraints: -------

$$f_{out}(i) + f_{det}(i) - \sum_j f_t(j,i) = 0$$

$$f_{in}(i) + f_{det}(i) - \sum_j f_t(j,i) = 0$$

$$f_{in}(i) + f_{det}(i) \in \{0,1\}$$

$$f_{out}(i) + f_{det}(i) \in \{0,1\}$$

$$f \in \{0,1\}$$

Summarize algebraically:

$$\boldsymbol{x} = [f_t(i,j) \dots, f_{in}(i), \dots, f_{out}(i), \dots, f_{det}(i), \dots]^T$$

$$\boldsymbol{c} = [C(i,j) \dots]^T$$

$$\mathcal{T}_* = \underset{\mathcal{T}}{\operatorname{argmin}}\, \boldsymbol{c}^T \boldsymbol{x}$$

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{0} \qquad , \boldsymbol{A} = \begin{bmatrix} 1,0,\dots,-1,-1\dots \\ \dots \end{bmatrix}$$

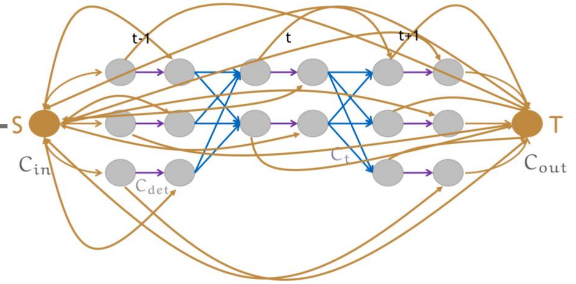
$$\boldsymbol{A_2}\boldsymbol{x} \in [\boldsymbol{0},\boldsymbol{1}] \qquad , \boldsymbol{A_2} = \begin{bmatrix} 1,1,0,\dots,0 \\ \dots \end{bmatrix}$$

$$\boldsymbol{x} \in [\boldsymbol{0},\boldsymbol{1}]$$

Good: Compact notation, feed to a solver

Bad: Binary optimization – NP-hard!!

Network flow formulation



- Binary optimization problem:

$$\mathcal{T}_* = \underset{\mathcal{T}}{\operatorname{argmin}}\, \boldsymbol{c}^T \boldsymbol{x}$$
$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{0}$$
$$\boldsymbol{A_2}\boldsymbol{x} \in [\boldsymbol{0}, \boldsymbol{1}]$$
$$\boldsymbol{x} \in [\boldsymbol{0}, \boldsymbol{1}]$$

$$\boldsymbol{A} = \begin{bmatrix} 1,1,\dots,-1,-1\dots \\ \dots \end{bmatrix}$$
$$\boldsymbol{A_2} = \begin{bmatrix} 1,1,0,\dots,0 \\ \dots \end{bmatrix}$$

NOTE (!):

- Matrices *A*,A_2 are unimodular (determinants of all square submatrices 1,0,-1)
- RHS are integral (all integer values)

- Can apply constraint relaxation and rewrite as a Linear Program (LP):

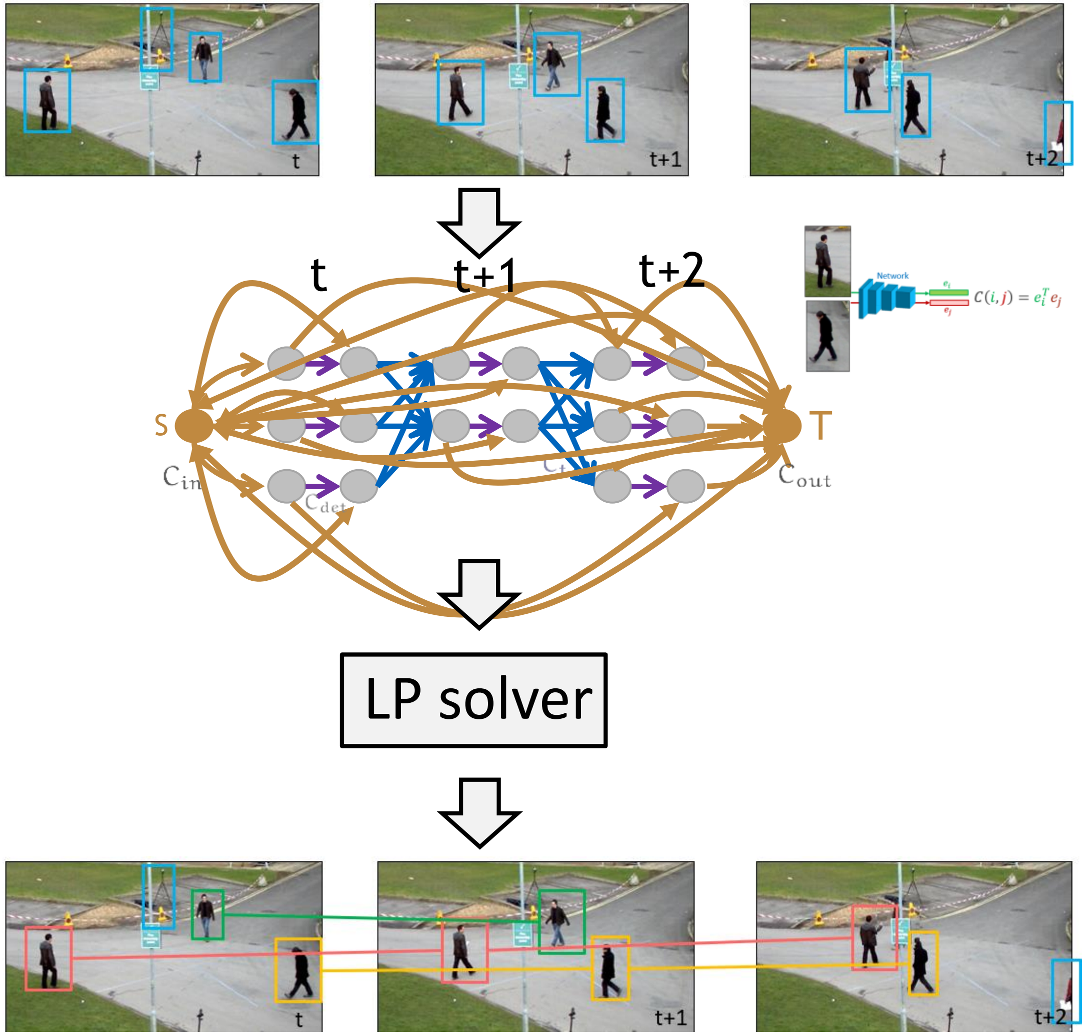
$$\mathcal{T}_* = \underset{\mathcal{T}}{\operatorname{argmin}}\, \boldsymbol{c}^T \boldsymbol{x}$$
$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{0}$$
$$\boldsymbol{0} \le \boldsymbol{A_2}\boldsymbol{x} \le \boldsymbol{1}$$
$$\boldsymbol{0} \le \boldsymbol{x} \le \boldsymbol{1}$$

Result:

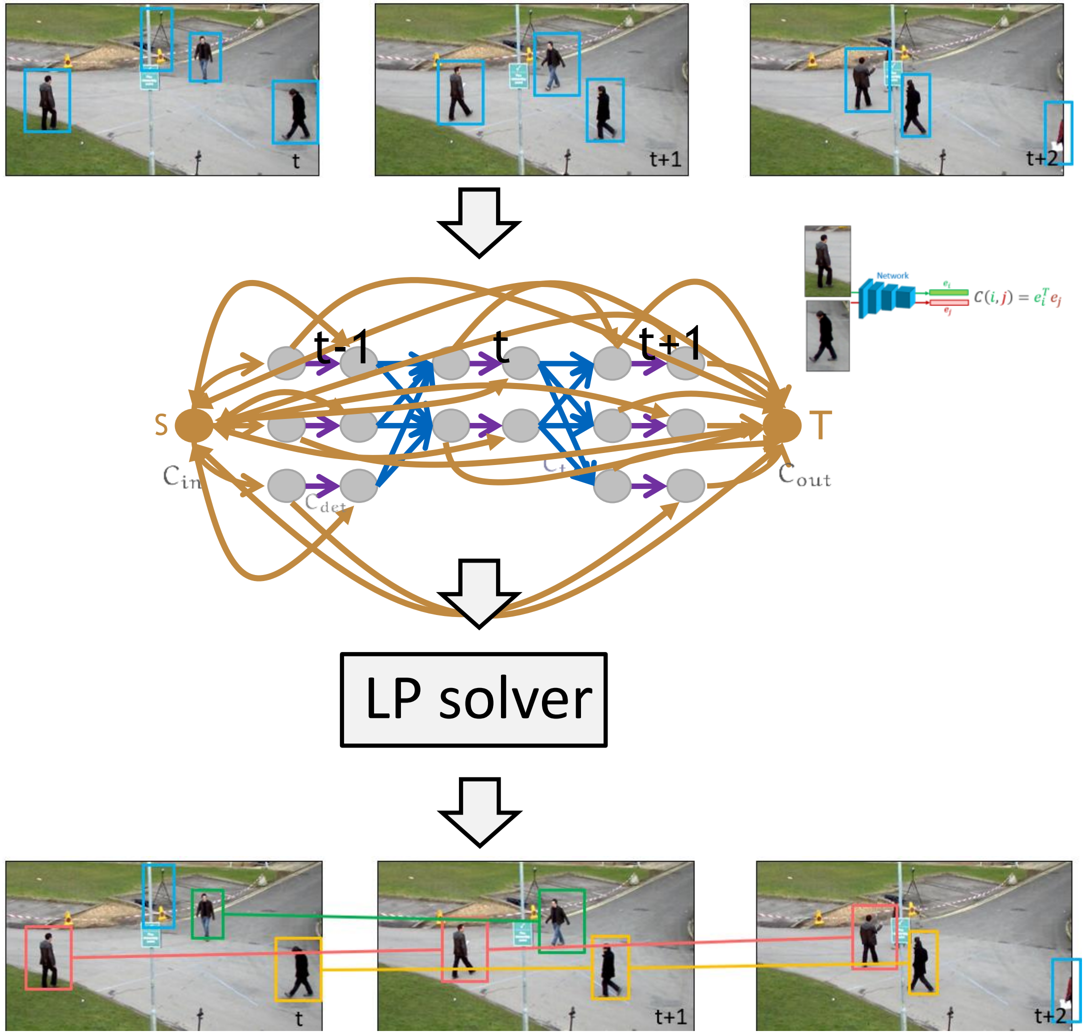
- Easily optimized with exiting tools (e.g., simplex)
- AND solutions guaranteed to be $\boldsymbol{x} \in [\boldsymbol{0}, \boldsymbol{1}]$!

Tracking with network flows: recap

- Run a detector on all frames
- Construct a graph among the detections using detection costs and between frame association costs (e.g., by visual similarity)
- Solve the Linear Program (e.g., by simplex)
- The output are binary activations on the graphs edges (i.e., associations).
- Apply post-processing if necessary (e.g., to re-connect broken trajectories)
- For very long sequences apply a sliding window



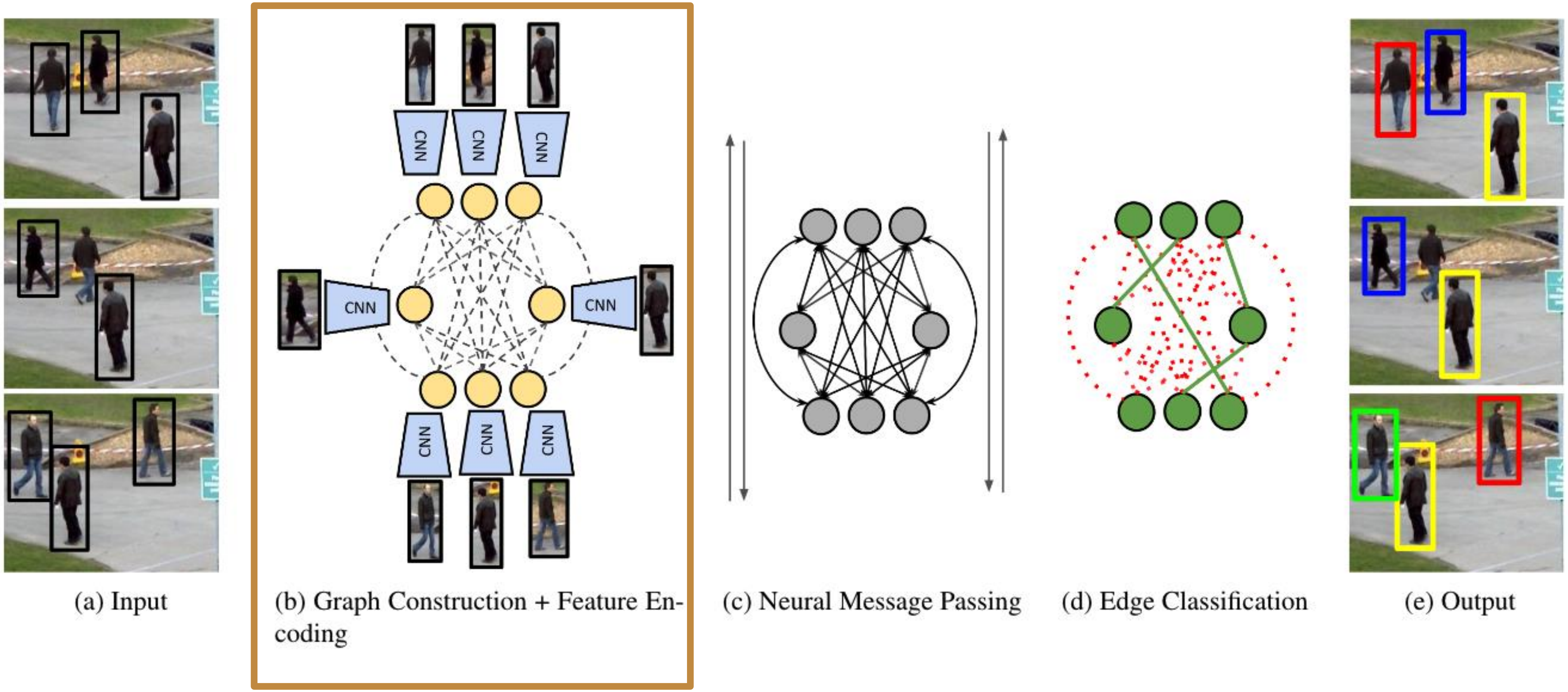
Tracking with network flows: Example



Hornakova, Henschel, Rosenhahn, Swoboda, Lifted Disjoint Paths with Application in Multiple Object Tracking, ICML2020

Learning the "solver" by graph neural nets

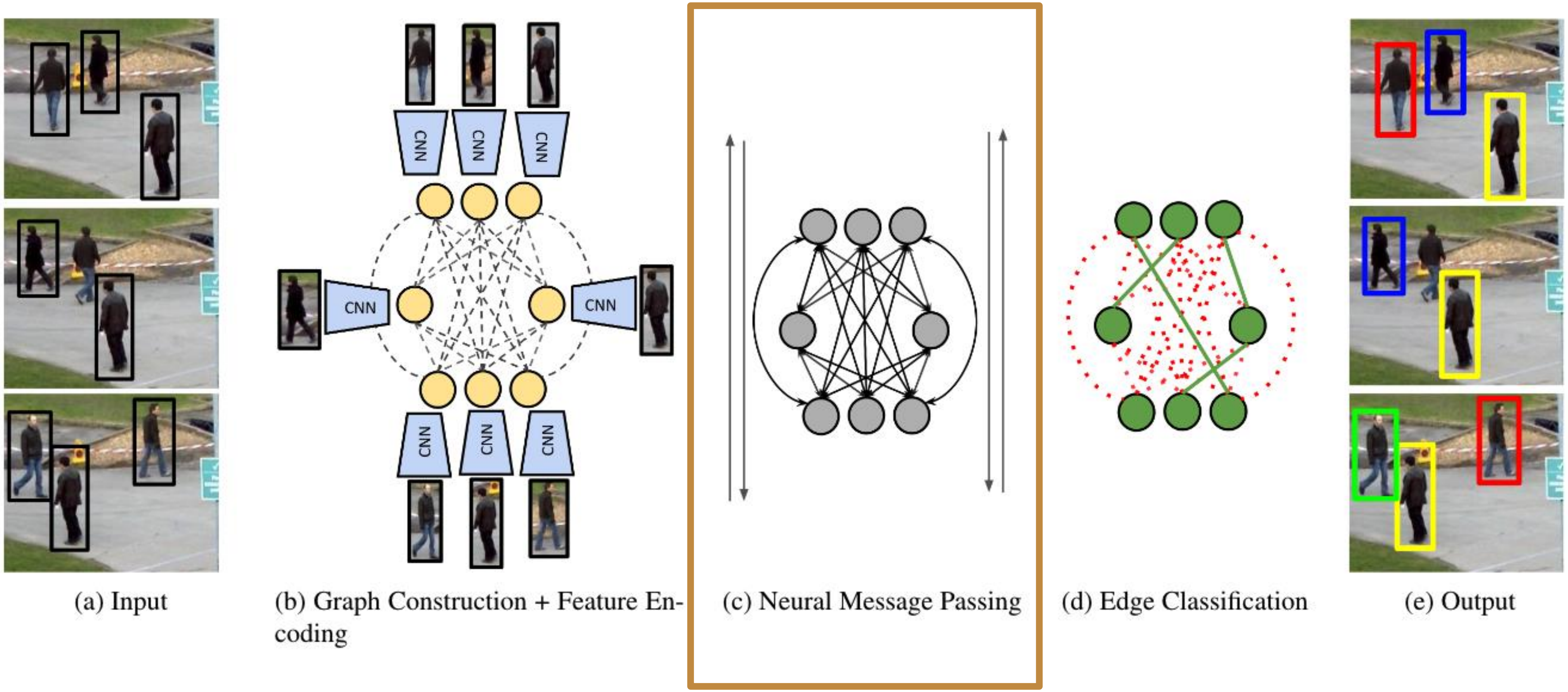
Encode appearance and scene geometry cues
into node and edge embeddings



G. Brasó and L. Leal-Taixé. "Learning a Neural Solver for Multiple Object Tracking", CVPR 2020, GIT

Learning the "solver" by graph neural nets

Propagate cues across the entire graph with neural message passing

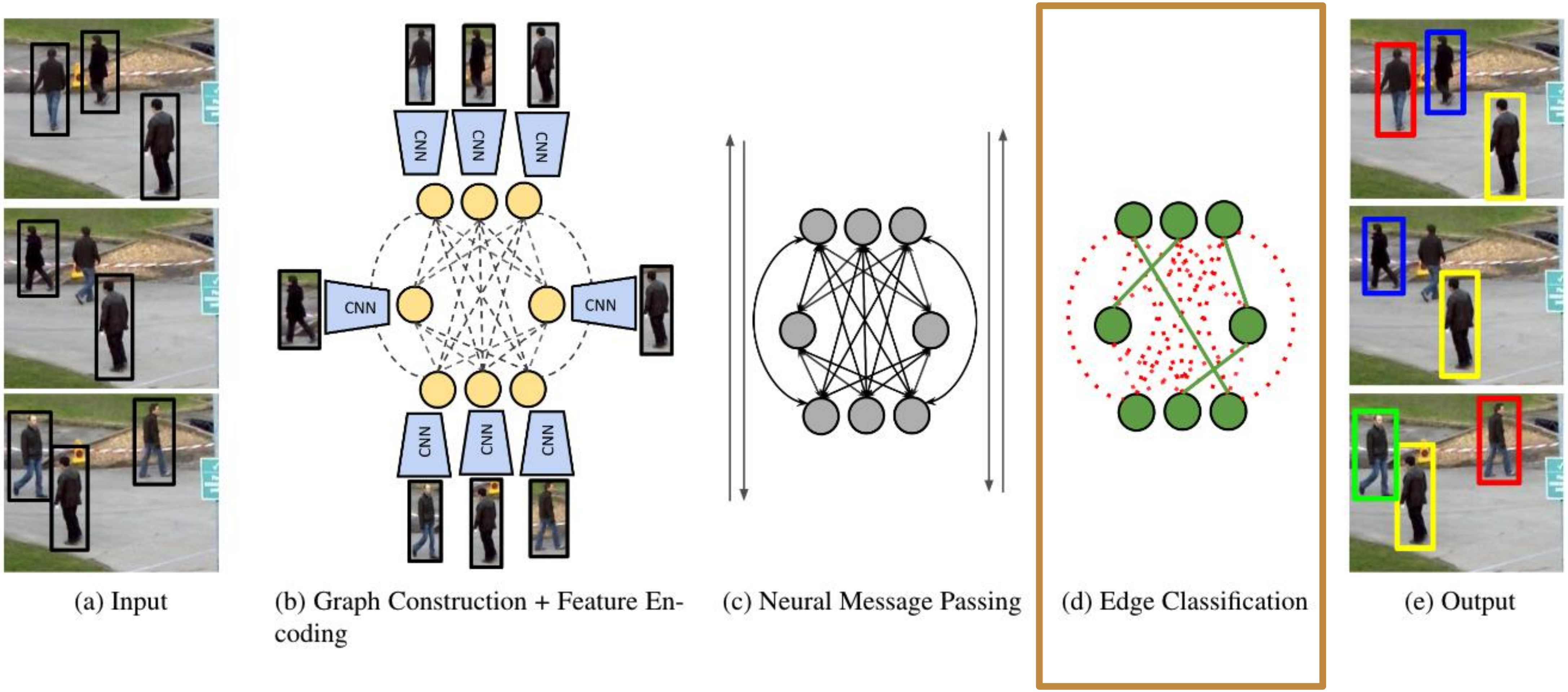


(a) Input (b) Graph Construction + Feature Encoding (c) Neural Message Passing (d) Edge Classification (e) Output

G. Brasó and L. Leal-Taixé. "Learning a Neural Solver for Multiple Object Tracking", CVPR 2020, GIT

Learning the "solver" by graph neural nets

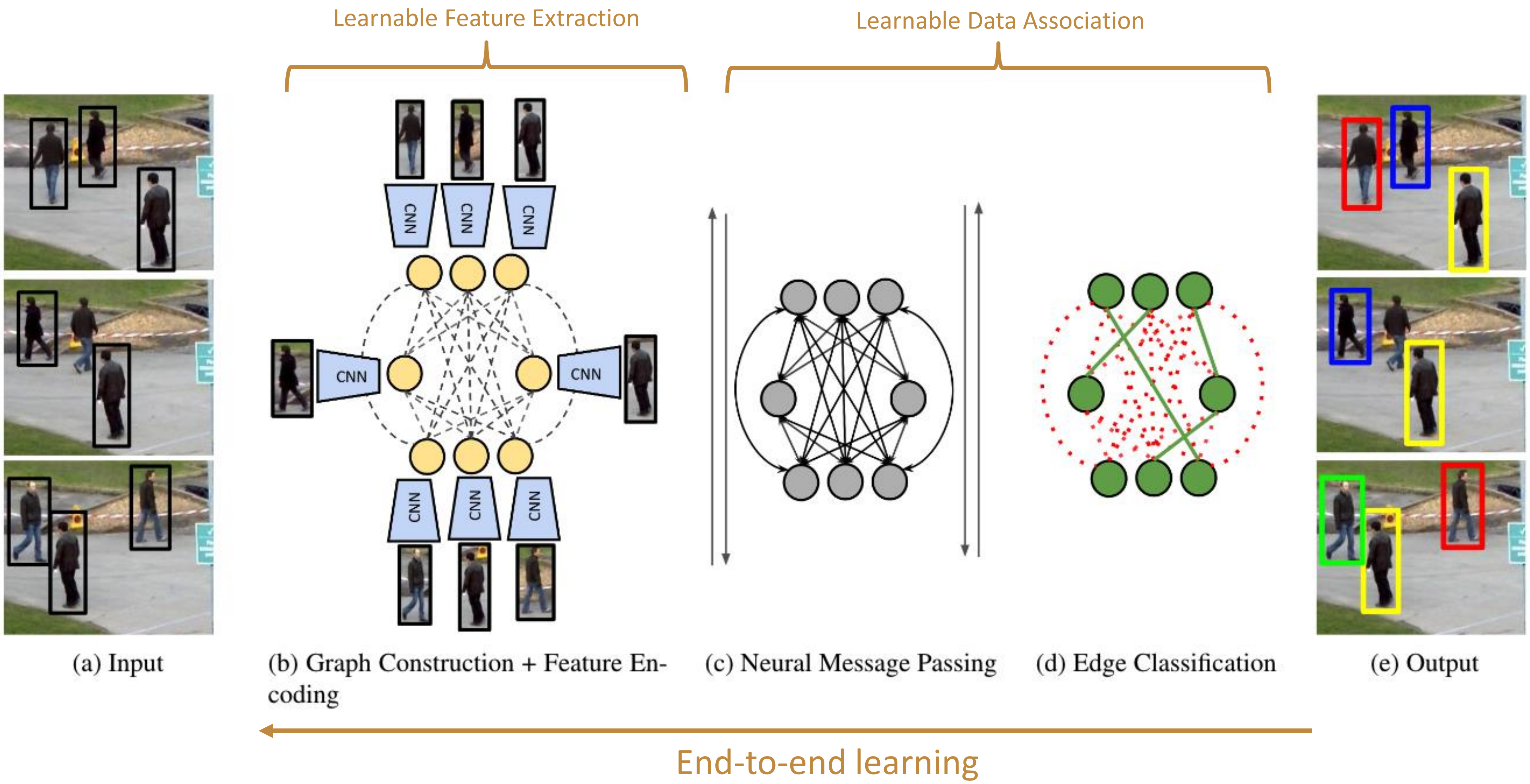
Learn to directly predict solutions of the Min-Cost Flow problem by classifying edge embeddings



(a) Input (b) Graph Construction + Feature Encoding (c) Neural Message Passing (d) Edge Classification (e) Output

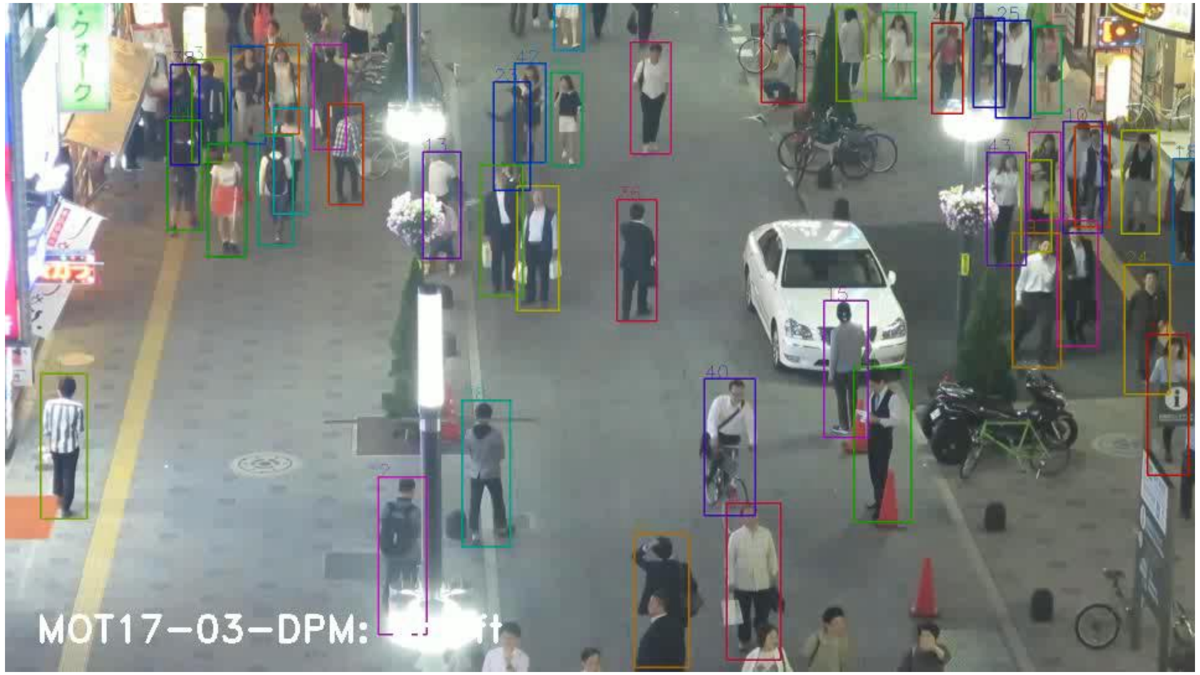
G. Brasó and L. Leal-Taixé. "Learning a Neural Solver for Multiple Object Tracking", CVPR 2020, GIT

Learning the “solver” by graph neural nets



G. Brasó and L. Leal-Taixé. “Learning a Neural Solver for Multiple Object Tracking”, CVPR 2020, GIT

Neural Message Passing Solver vs Lifted Paths



Hornakova, Kaiser, Rolinek, Rosenhahn, Swoboda, Henschel. Making Higher Order MOT Scalable: An Efficient Approximate Solver for Lifted Disjoint Paths, ICCV 2021.

G. Braso, L. Leal-Taixe. Learning a Neural Solver for Multiple Object Tracking. CVPR, 2020.

Results from: https://motchallenge.net

Single-target tracking cast in an MOT framework

Yan et al. “Layered Data Association Using Graph-Theoretic Formulation with Application to Tennis Ball Tracking in Monocular Sequences”, IEEE TPAMI, 2008

Online vs Batch Multiple Object Tracking

- Batch tracking considers "all" frames to infer position at t
  (useful for offline applications, e.g., post-hoc analysis, video editing)

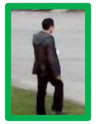
- Online tracking consider only frames before t to infer position at t
  (useful in realtime applications, e.g., drones)

Online tracking: Frame-to-frame tracking

- Frame-to-frame tracking is the most basic form of online tracking

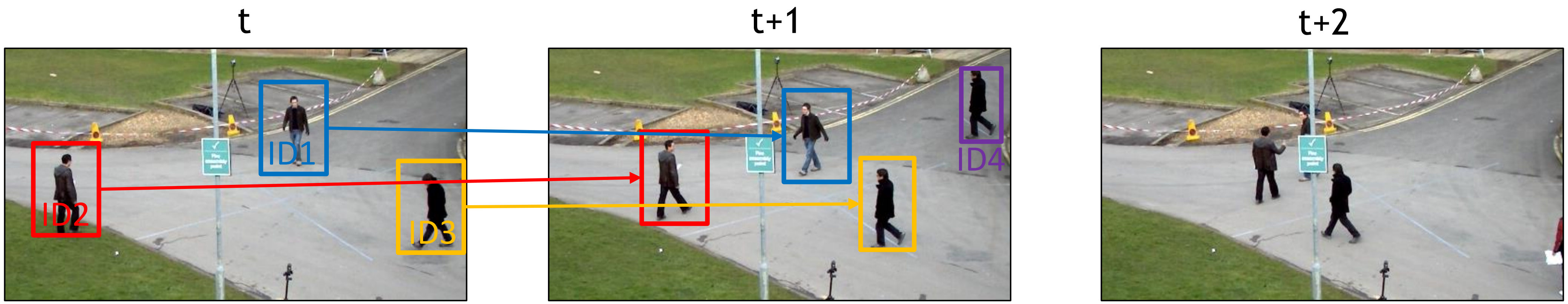
t t+1 t+2

A Tracking iteration steps:

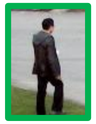
1. Run a detector  on each new frame *t+1*

Online tracking: Frame-to-frame tracking

- Frame-to-frame tracking is the most basic form of online tracking

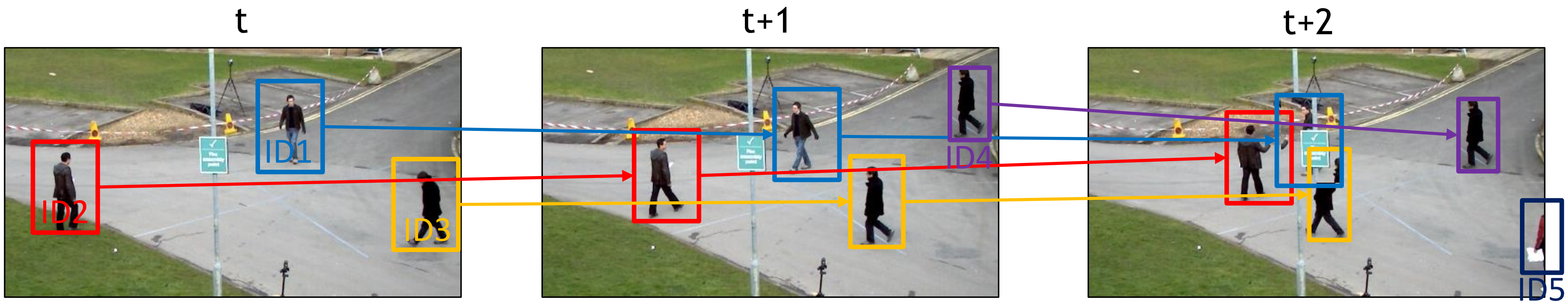


A Tracking iteration steps:

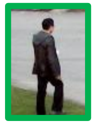
1. Run a detector  on each new frame *t+1*
2. Match detections from frame *t* with the detections at frame *t+1*
3. Initialize/kill tracks (i.e., new objects entering, existing leaving)

Online tracking: Frame-to-frame tracking

- Frame-to-frame tracking is the most basic form of online tracking

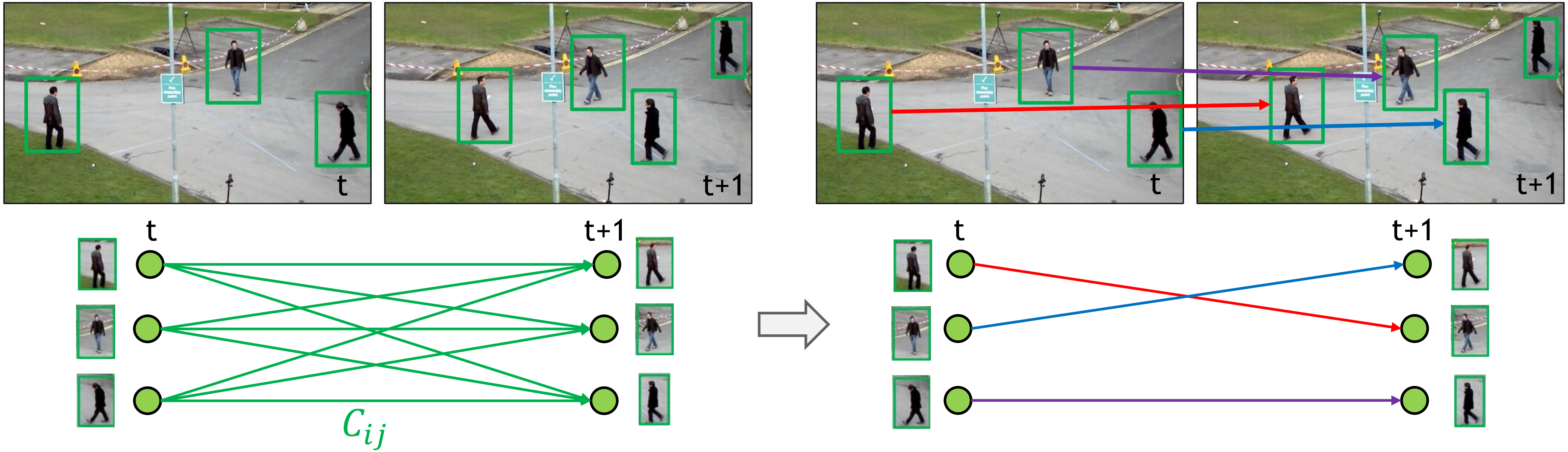


A Tracking iteration steps:

1. Run a detector  on each new frame *t+1*
2. Match detections from frame *t* with the detections at frame *t+1*
3. Initialize/kill tracks (i.e., new objects entering, existing leaving)
4. Repeat from step (1) for all consecutive frames

Frame-to-frame matching

- Bipartite graph matching formulation:



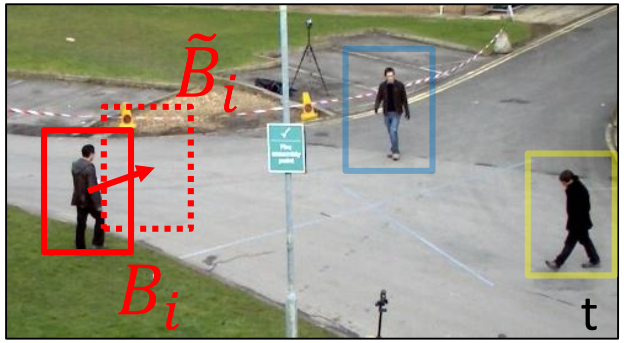
1. Create a graph of all possible assignments between two frames (with costs)
2. Find a one-to-one matching solution by minimizing the total cost

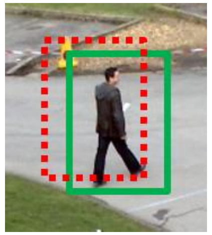
- Q1: What should be the cost? Q2: How to solve the bipartite matching?

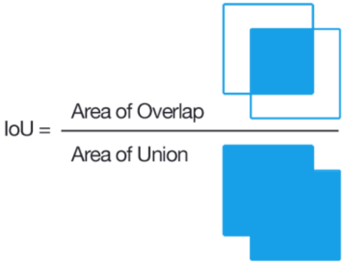
The assignment costs C_{ij}

- Motion-based similarity:
  IoU between the detection and predicted bounding box by Kalman filter.
- Appearance-based similarity:
  Dot product between visual embeddings.
- Assignment cost:

$$C_{ij} = \begin{cases} \lambda c_{i,j}^{(M)} + (1-\lambda) c_{i,j}^{(A)} \\ 0\ ; c_{i,j}^{(M)} > \theta_M \vee c_{i,j}^{(A)} > \theta_A \end{cases}$$

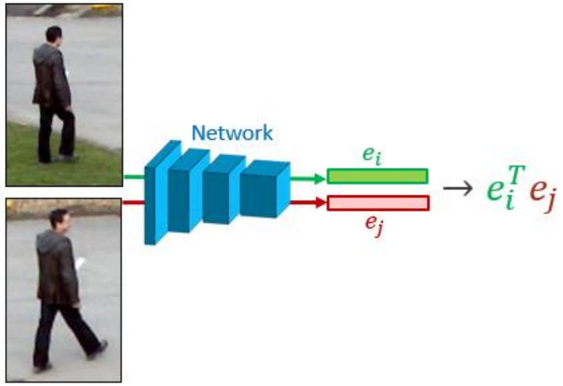






Motion-based cost:

$$c_{i,j}^{(M)} = 1 - IoU(\tilde{B}_i, B_j)$$

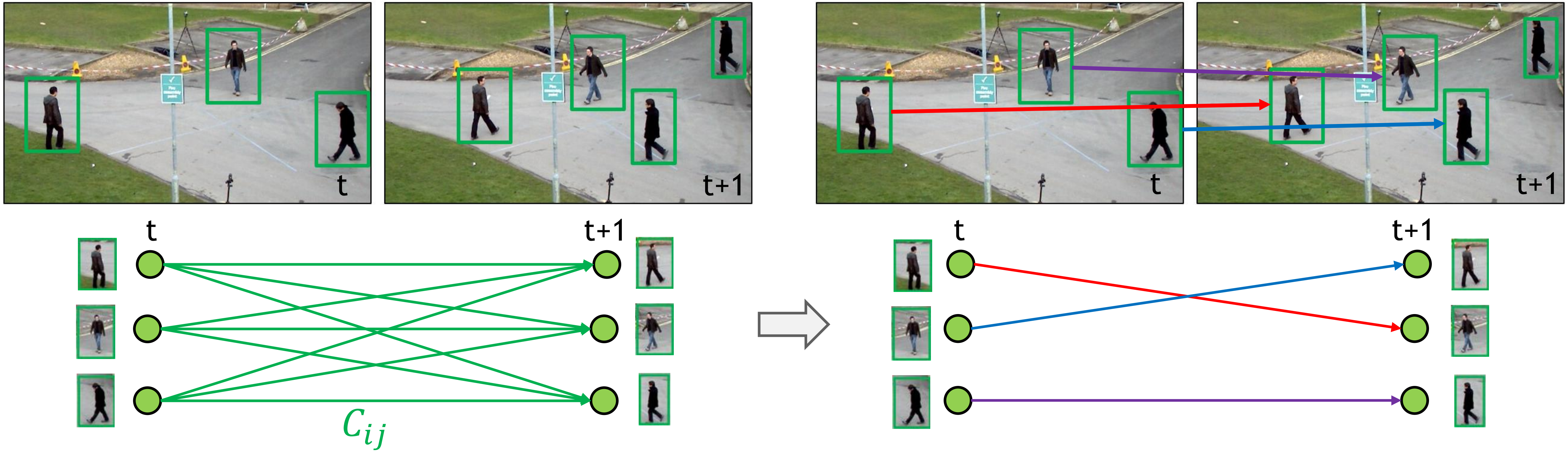


Appearance-based cost:

$$c_{i,j}^{(A)} = 1 - \boldsymbol{e}_i^T \boldsymbol{e}_j$$

Frame-to-frame matching

- Bipartite graph matching formulation:



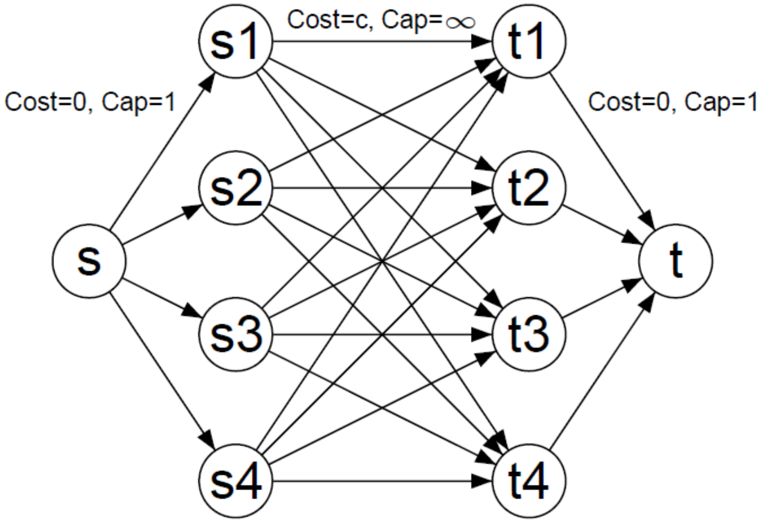
1. Create a graph of all possible assignments between two frames (with costs)
2. Find a one-to-one matching solution by minimizing the total cost

- Q1: What should be the cost? Q2: How to solve the bipartite matching?

Bipartite matching

[1] Demo: http://www.hungarianalgorithm.com/solve.php

- For each box at *t*, find a unique match in *t+1* and vice versa.
- Special case of graph matching by LP



- Efficient algorithm exists:

  [1]Hungarian algorithm

$$x_{opt} = \underset{x}{\operatorname{argmin}}\, c^T x$$

Write costs into a matrix:

Predictions

Detections

	0.9	0.8	0.8	0.1
	0.5	0.4	0.3	0.8
	0.2	0.1	0.4	0.8
	0.1	0.2	0.5	0.9

Bipartite matching

[1] Demo: http://www.hungarianalgorithm.com/solve.php

- Solution:

$$\boldsymbol{x}_{opt} = \underset{\boldsymbol{x}}{\operatorname{argmin}}\, \boldsymbol{c}^T \boldsymbol{x}$$
$$\boldsymbol{x} \in \{0,1\}$$

Unique assignments $\boldsymbol{x}_{opt}$ that minimize the total cost (sum of costs).

Write costs into a matrix:

Predictions

Detections

0.9	0.8	0.8	0.1
0.5	0.4	0.3	0.8
0.2	0.1	0.4	0.8
0.1	0.2	0.5	0.9

Bipartite matching

[1] Demo: http://www.hungarianalgorithm.com/solve.php

- What happens if we have missing prediction?

Write costs into a matrix:

Predictions

Detections				
	0.9	0.8	0.8	
	0.5	0.4	0.3	
	0.2	0.1	0.4	
	0.1	0.2	0.5	

Bipartite matching

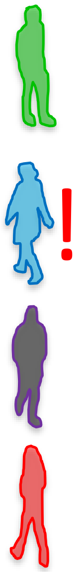
[1] Demo: http://www.hungarianalgorithm.com/solve.php

- What happens if we have missing prediction?
- What happens if there are many predictions, but some detections are not within the list of predictions?

Write costs into a matrix:

Predictions

Detections



!

0.9	0.8	0.8	
0.5	0.4	0.7	
0.2	0.1	0.4	
0.1	0.2	0.5	

Note: Incorrect assignment

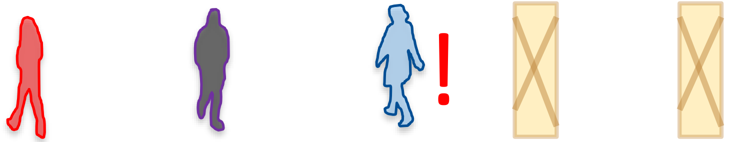
Bipartite matching

[1] Demo: http://www.hungarianalgorithm.com/solve.php

- What happens if we have missing prediction?
- What happens if there are many predictions, but some detections are not within the list of predictions?
- Introduce extra nodes that act as threshold on the acceptable assignment cost

Write costs into a matrix:

Predictions

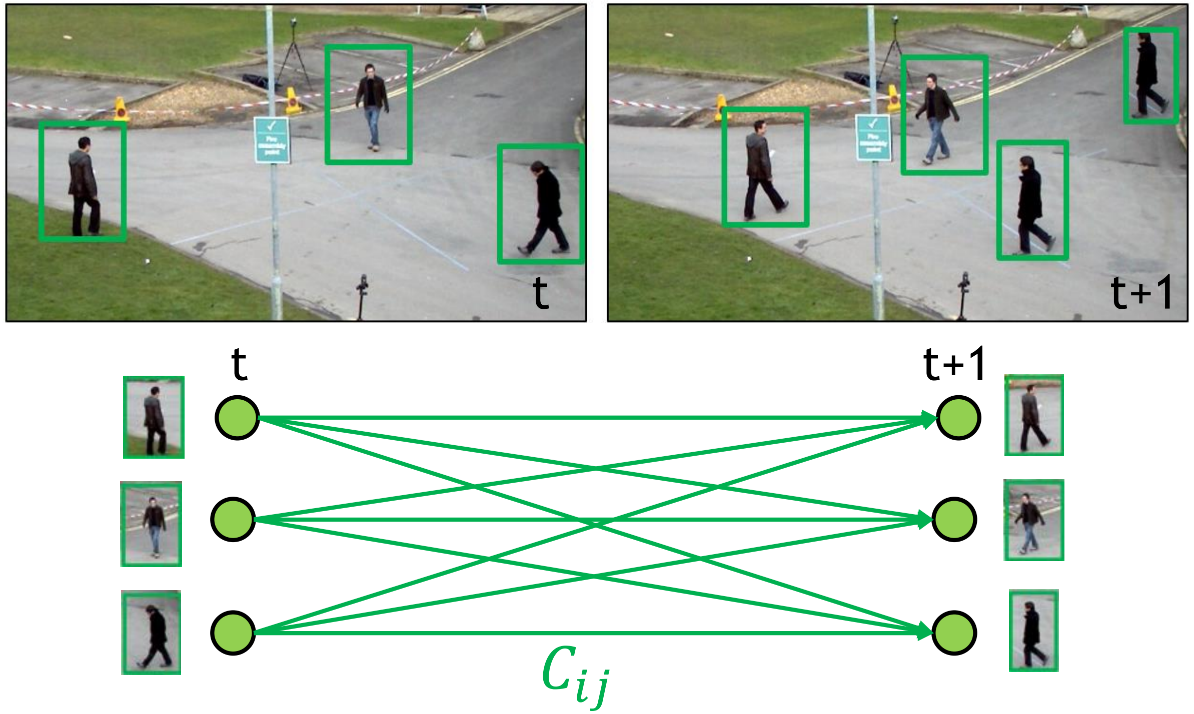


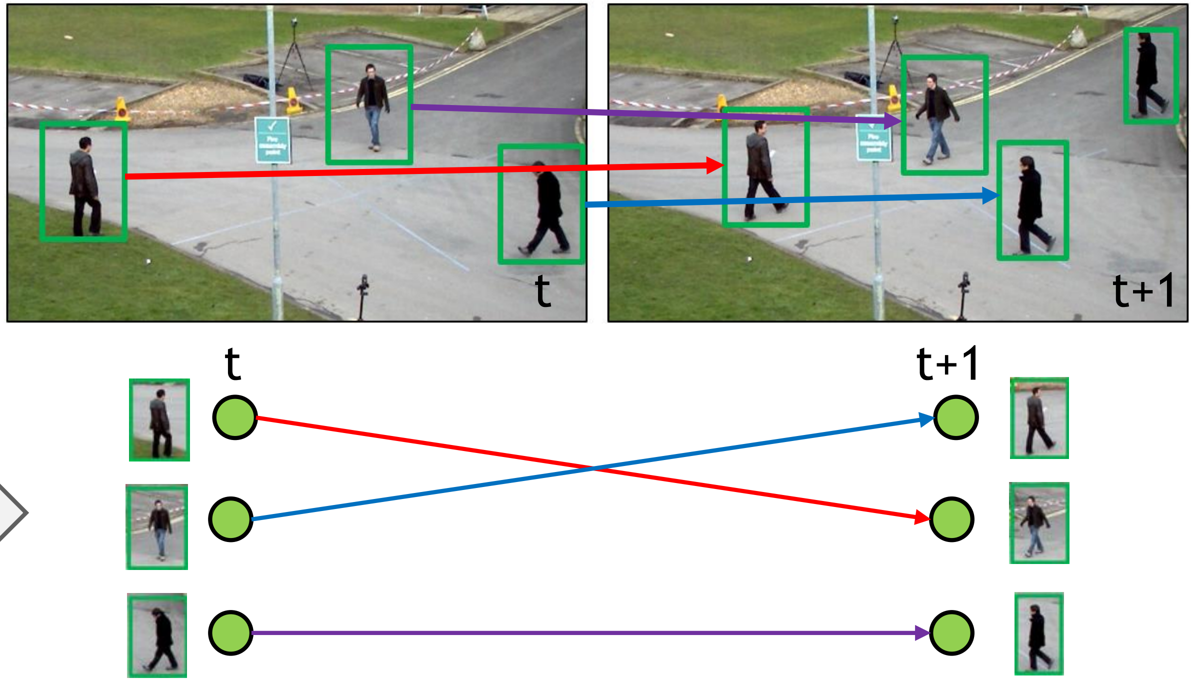
Detections

0.9	0.8	0.8	0.3	0.3
0.5	0.4	0.7	0.3	0.3
0.2	0.1	0.4	0.3	0.3
0.1	0.2	0.5	0.3	0.3
0.3	0.3	0.3	0.3	0.3

Frame-to-frame matching

- Bipartite graph matching formulation:



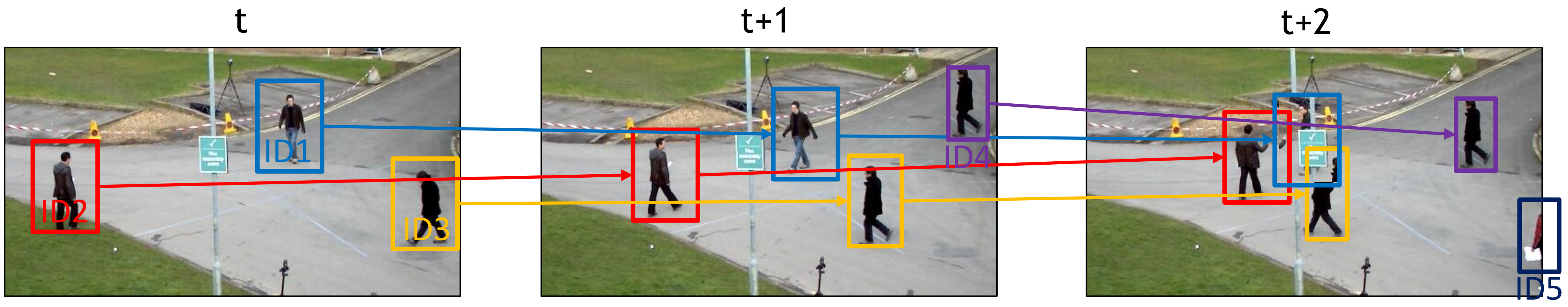


1. Create a graph of all possible assignments between two frames (with costs)
2. Find a one-to-one matching solution by minimizing the total cost

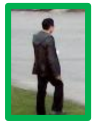
- Q1: What should be the cost? Q2: How to solve the bipartite matching?

Online tracking: Frame-to-frame tracking

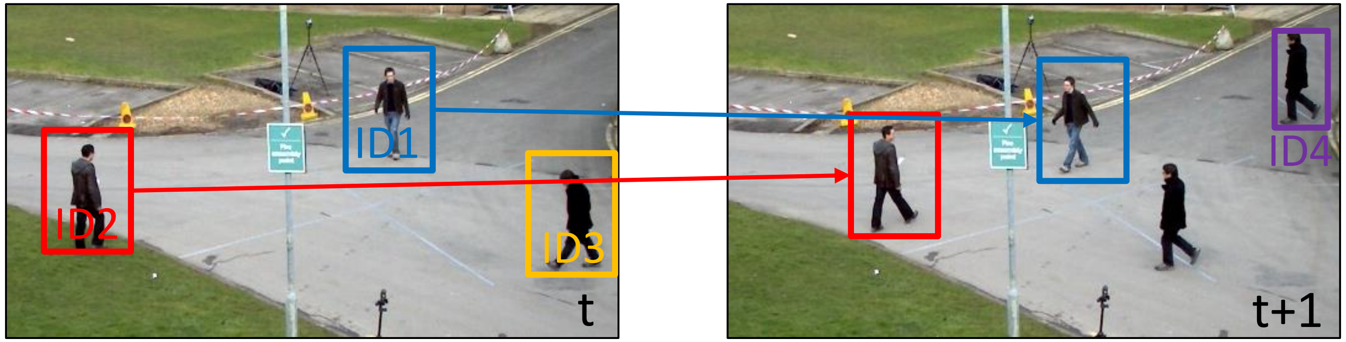
- Frame-to-frame tracking is the most basic form of online tracking



A Tracking iteration steps:

1. Run a detector  on each new frame *t+1*
2. Match detections from frame *t* with the detections at frame *t+1*
3. Initialize/kill tracks (i.e., new objects entering, existing leaving)
4. Repeat from step (1) for all consecutive frames

Track management (initalization / termination)



Classical example: DeepSORT

Wojke et al., Simple Online and Realtime Tracking with a Deep Association Metric, ICIP 2017

Managing unmatched predictions (example ID3):

- Continue predicting by Kalman filter in the next frames.
- If not matched within the next T_{max} frames, terminate the trajectory.

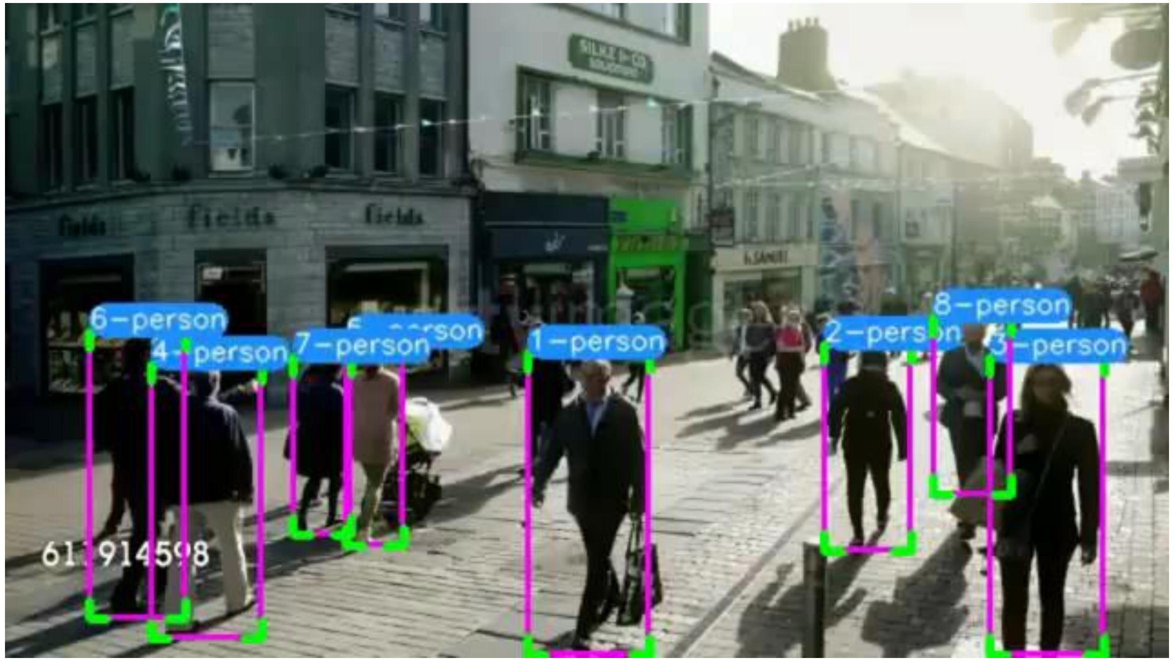
Managing unmatched detections (example ID4):

- If detection score high, initialize a new track (new ID) and enter a probation period.
- If successfully matched for T_{min} consecutive frames, accept as a track.

A simple, but more advanced MOT: BYTE

- Main idea: do not immediately discard detections with low detection score, but rather schedule the matching pipeline.
  - First match the tracks with highly-scored detections,
  - then match the unmatched tracks with the remaining detections.

https://www.youtube.com/watch?v=ehKj7mdISL0

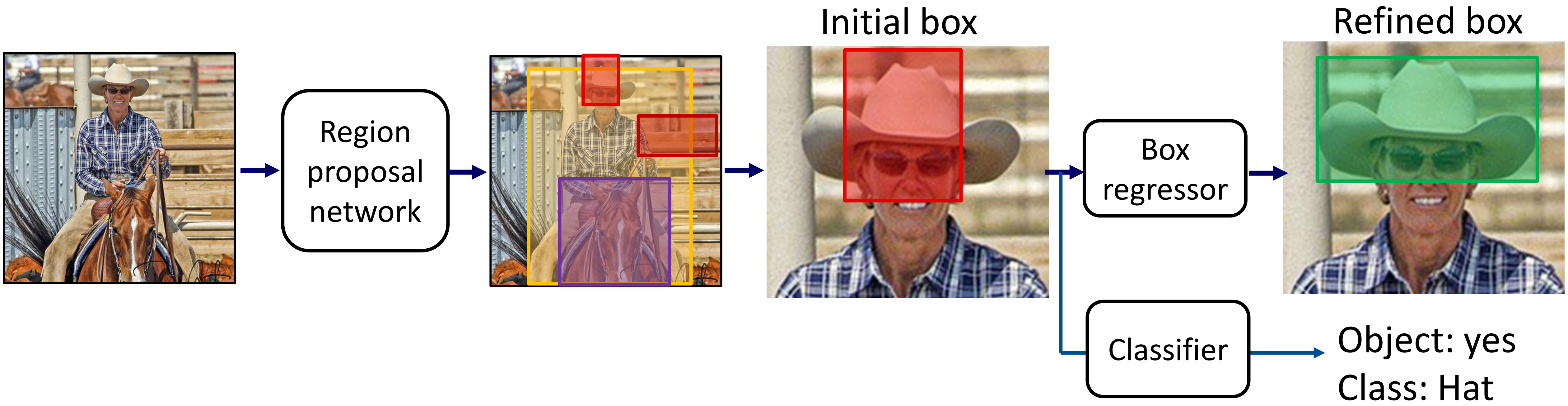


https://www.youtube.com/watch?v=ET6AMGniN2Y

Zhang et al., ByteTrack: Multi-Object Tracking by Associating Every Detection Box, ECCV 2022, GIT

Tracktor(++)[2]

- Recall R-CNN pipeline for object detection:
  - Propose regions
  - Refine and classify each region
- Idea: Exploit RCNN[1] architecture as a detector & tracker



[1]Ren, He, Girshick, Sun. Faster R-CNN: Towards Real-Time Object Detection. NIPS 2015.

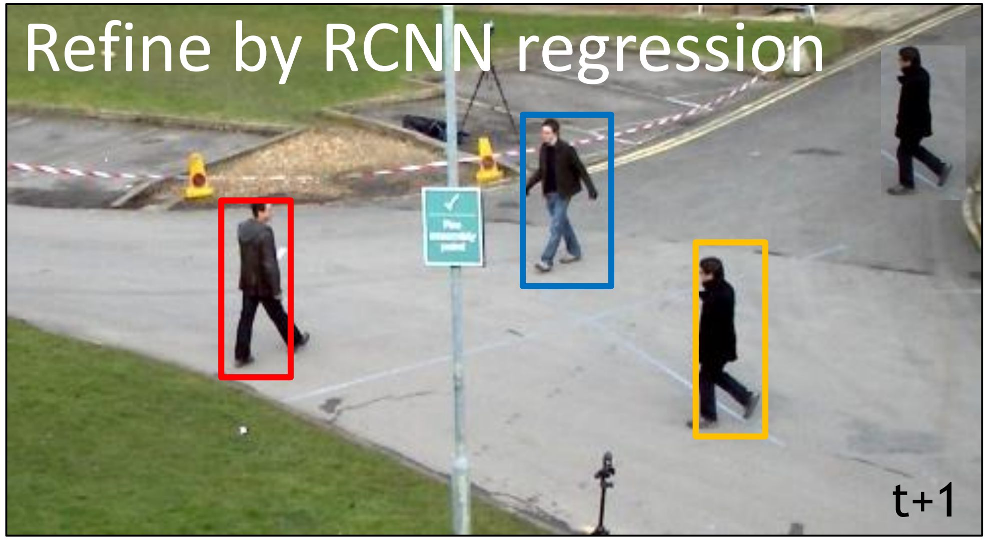
[2] P. Bergmann, T. Meinhardt, L. Leal-Taixé. Tracking without bells and whistles. In ICCV, 2019

Tracktor(++)

- Predict each bounding box from *t* by a Kalman filter
- On image *t+1*, improve each translated box by the RCNN refinement head
- Run an RCNN detector to detect potential objects
- Associate the predicted boxes with detections & create new tracks

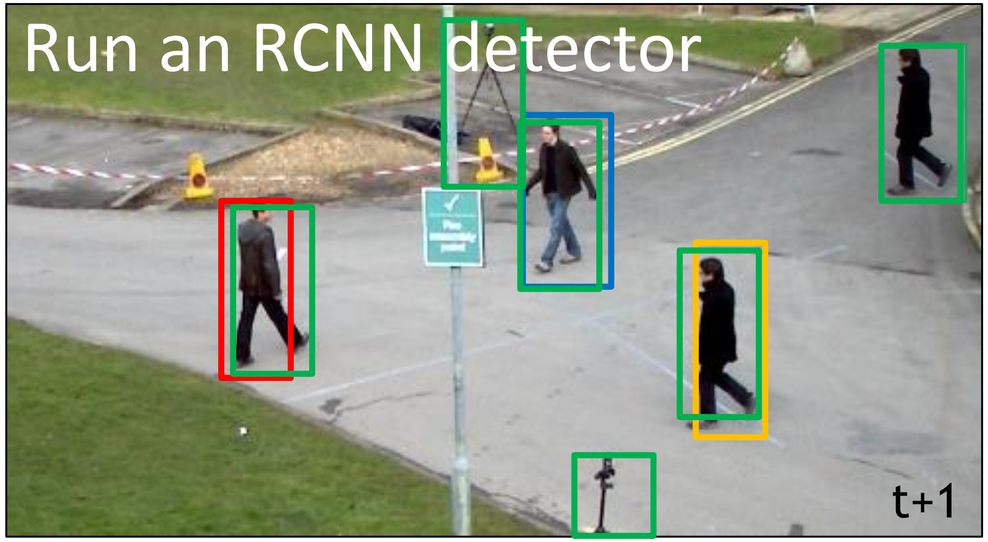
Tracktor(++)

- Predict each bounding box from *t* by a Kalman filter
- On image *t+1*, improve each translated box by the RCNN refinement head
- Run an RCNN detector to detect potential objects
- Associate the predicted boxes with detections & create new tracks



Tracktor(++)

- Predict each bounding box from *t* by a Kalman filter
- On image *t+1*, improve each translated box by the RCNN refinement head
- Run an RCNN detector to detect potential objects
- Associate the predicted boxes with detections & create new tracks

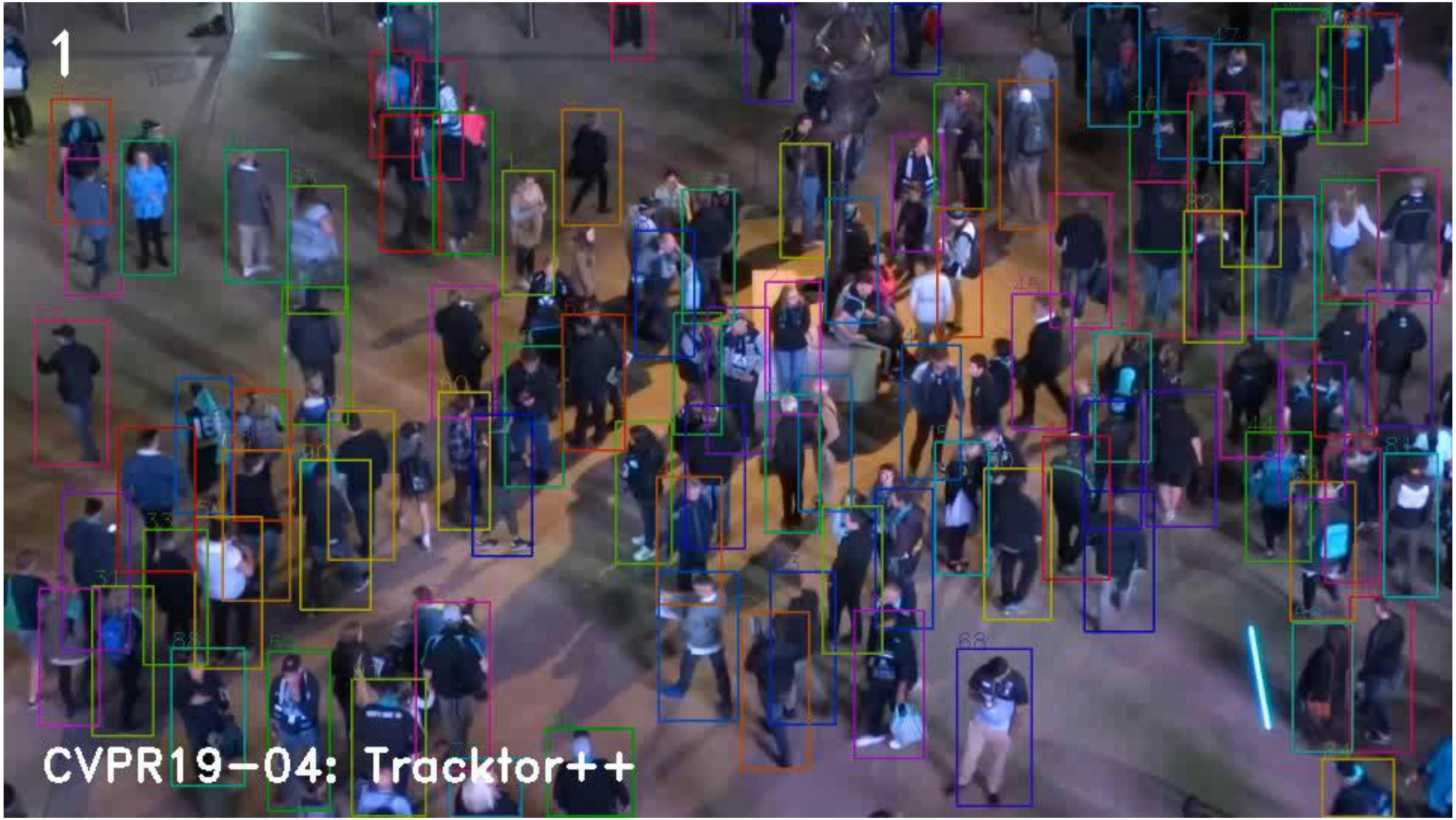


Tracktor(++)

- Predict each bounding box from *t* by a Kalman filter
- On image *t+1*, improve each translated box by the RCNN refinement head
- Run an RCNN detector to detect potential objects
- Associate the predicted boxes with detections & create new tracks

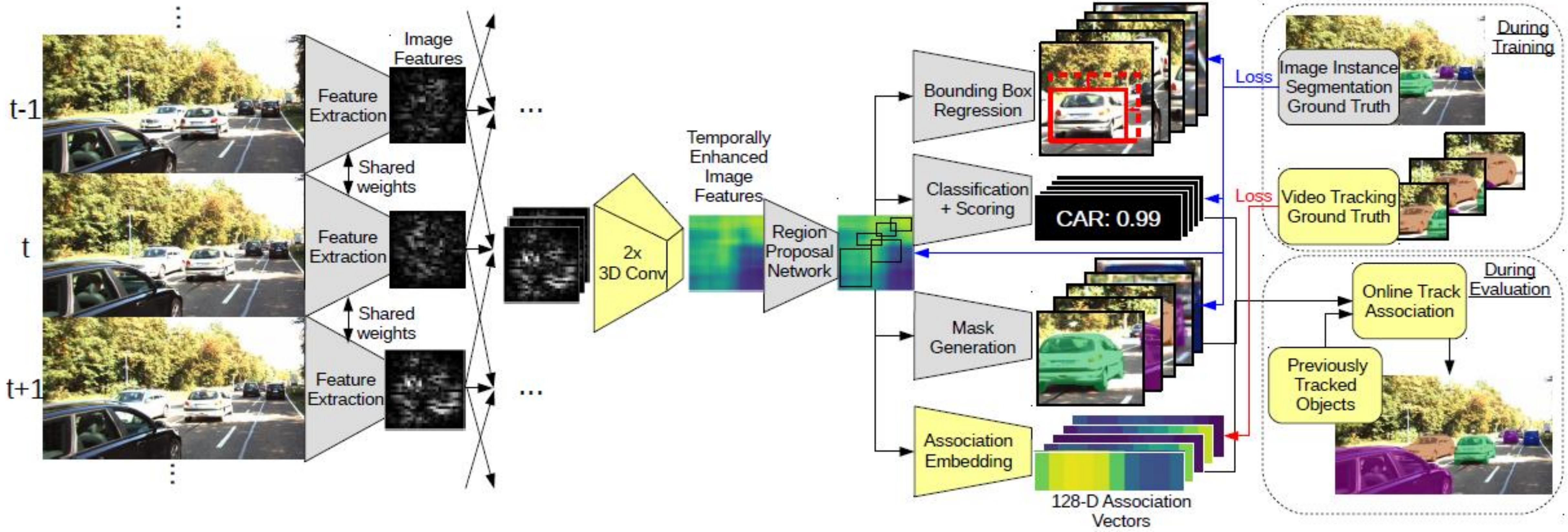
(Image stabilisation used to compensate for large between-frame camera motions.)

Tracktor++



Multi-object tracking and segmentation

- Learns detection, segmentation, association embeddings



Paul Voigtlaender and Michael Krause and Aljosa Osep and Jonathon Luiten and Berin Ba, MOTS: Multi-Object Tracking and Segmentation, CVPR2019

Multi-object tracking and segmentation

Paul Voigtlaender and Michael Krause and Aljosa Osep and Jonathon Luiten and Berin Ba, MOTS: Multi-Object Tracking and Segmentation, CVPR2019, [GIT]

Further reading & resources

- MOTChallenge: www.motchallenge.net (people)
  - Several challenges from less to more crowded

- KITTI benchmark: http://www.cvlibs.net/datasets/kitti/ (vehicles)
- UA-Detrac: http://detrac-db.rit.albany.edu (vehicles)
- Papers with code: https://paperswithcode.com/task/multi-object-tracking

Acknowledgment

Many thanks to Laura Leal Taixe and Aljoša Ošep for generously sharing their slides that ended up in this presentation and for loads of inspiration.

Check out Laura’s awesome lectures on Youtube:

CV3DST - Computer Vision 3

Further reading & resources

- End-to-end learnable trackers: Famnet
- Tracking by segmentation: Tracking everything & segmenting

  Bastian Leibe Aljoša Ošep